DRIVER MANUAL PMC-990799



C. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

PM7350



S/UNI-DUPLEX

DUAL SERIAL LINK, PHY MULTIPLEXER

DRIVER MANUAL

PRELIMINARY

ISSUE 1: JULY 1999

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL



DRIVER MANUAL PMC-990799



ra, Inc. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

REVISION HISTORY

Issue No.	Issue Date	Originator	Details of Change
Issue 1	July 1999	James Lamothe	Document created from S/UNI-DUPLEX Driver Design Spec (PMC-981033 Issue 2) and the S/UNI-VORTEX Driver Manual (PMC-990786 Issue 1)

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

ABOUT THIS MANUAL

This manual describes the S/UNI-DUPLEX device driver. It describes the driver's functions, data structures, and architecture. This manual focuses on the driver's interfaces to your application, real-time operating system, and to the S/UNI-DUPLEX device. It also describes in general terms how to modify and port the driver to your software and hardware platform.

Audience

This manual was written for people who need to:

- Evaluate and test the S/UNI-DUPLEX device
- Modify and add to the S/UNI-DUPLEX driver's functions
- Port the S/UNI-DUPLEX driver to a particular platform.

References

For more information about the S/UNI-DUPLEX driver, see the driver's release notes. For more information about the S/UNI-DUPLEX device, see the following documents:

- S/UNI-DUPLEX (Dual Serial Link, Phy Multiplexer) Datasheet: PMC-980581
- S/UNI-DUPLEX (Dual Serial Link, Phy Multiplexer) Short Form Datasheet: PMC-990174)
- S/UNI-DUPLEX and S/UNI-VORTEX Technical Overview: PMC-981025

Note: Ensure that you use the document that PMC_Sierra issued for your version of the device and driver.

DRIVER MANUAL PMC-990799 PMC-Sierra, Inc.

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

TABLE OF CONTENTS

1	Drive	r Porting	Quick Start	.13
2 Driver Functions and Features		ons and Features	.15	
	2.1	Driver A 2.1.1 2.1.2 2.1.3 2.1.4 2.1.5 2.1.6 2.1.7	Architecture Driver API Module Driver Real-Time-OS Interface Module Driver Hardware-Interface Module Driver Library Module Device Data-Block Module Interrupt-Service Routine Module Deferred-Processing Routine Module	.16 .17 .18 .18 .18 .18 .19 .19
	2.2	Driver S	Software States	.19
	2.3	Process 2.3.1 2.3.2 2.3.3 2.3.4	sing Flows Device Initialization, Re-initialization, and Shutdown Cell Extraction Interrupt Servicing Polling Servicing	.20 .21 .22 .23 .26
3	Drive	r Data Si	tructures	.29
	3.1	Cell Dat 3.1.1 3.1.2	ta Structures Cell-Header Data Structure Cell-Control Data Structure	.29 .29 .29
	3.2	Device- 3.2.1 3.2.2	Configuration Data Structures Initialization Data Structure Register Data Structure	.30 .30 .31
	3.3	Device- 3.3.1 3.3.2	Context Data Structures Global Driver-Database Structure Device Data-Block Structure	.33 .33 .34
	3.4	Interrup 3.4.1 3.4.2	ot Data Structures Interrupt-Enable Data Structure Interrupt-Context Data Structure	.36 .36 .37
	3.5	Count S 3.5.1 3.5.2	Structures HSS Counts Statistical Counts	.38 .38 .38
4	Applic	cation In	terface Functions	.43
	4.1	Driver Iı 4.1.1 4.1.2	nitialization and Shutdown duplexModuleInit: Initializing Driver Modules duplexModuleShutdown: Shutting Down Driver Modules	.44 .44 .45

PMC-990799

DRIVER MANUAL

PMC-Sierra, Inc.

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4.2	Device Addition, Reset, and Deletion 4.2.1 duplexAdd: Adding Devices 4.2.2 duplexReset: Resetting Devices 4.2.3 duplexRemoteReset: Resetting Other Devices 4.2.4 duplexDelete: Deleting Devices	45 45 46 47 47
4.3	Reading from and Writing to Devices	48 48 49
4.4	Device Initialization4.4.1duplexInit: Initializing Devices4.4.2duplexInstallIndFn: Installing Indication Callback Functions4.4.3duplexRemoveIndFn: Removing Indication Callback Functions	49 49 50 51
4.5	Device Activation and Deactivation 4.5.1 duplexActivate: Activating Devices 4.5.2 duplexDeactivate: Deactivating Devices	52 52 52
4.6	Device Diagnostics4.6.1duplexRegisterTest: Verifying Device Register Access4.6.2duplexLoopback: Enabling/Disabling Diagnostic or Line Loopback4.6.3duplexGetClockStatus: Monitoring Device Clocks	53 53 54 55
4.7	 HSS Link Configuration	56 56 57 57 58
4.8	 HSS-Link Cell Insertion and Extraction 4.8.1 duplexInsertCell: Inserting Cells into HSS Links 4.8.2 duplexExtractCell: Extracting Cells from HSS Links 4.8.3 duplexCheckExtractFifos: Getting Contents of the Extract-FIFO-Ready Register 4.8.4 duplexEnableRxCellInd: Enabling the Received Cell Indicator 4.8.5 duplexInstallCellTypeFn: Installing Callback Functions 	59 60 61 63 64 64
4.9	BOC Transmission and Reception 4.9.1 duplexTxBOC: Transmitting BOC 4.9.2 duplexRxBOC: Reading from Received BOC 4.9.3 duplexSetAutoRDITx: Transmitting Remote-Defect Indication Code Words 4.9.4 duplexSciAnyPhyGetConfig: Getting SCI-PHY/Any-PHY Configuration Informatio 4.9.5 duplexSciAnyPhySetConfig: Configuring HSS-Links	65 65 66 67 n68 70
4.10	Clocked Serial-Data Interface Functions 4.10.1 duplexRxSerChnlReadReg: Reading from Receive Serial-Channel Context Bytes 4.10.2 duplexRxSerChnlWriteReg: Writing to Receive Serial-Channel Context Bytes 4.10.3 duplexRxSerChnlHCSCntResetEn: Enabling Auto Reset of HCS Error Registers. 4.10.4 duplexTxSerChnlReadReg: Reading from Transmit Serial-Channel Context Bytes 4.10.5 duplexTxSerChnlWriteReg: Writing to Transmit Serial-Channel Context Bytes	71 71 72 73 74 75

PMC	PMC-Sierra, Inc.
-----	------------------

ISSUE 1

PM7350 S/UNI-DUPLEX DRIVER

DRIVER MANUAL PMC-990799

S/UNI-DUPLEX DRIVER MANUAL

4.11.1 duplexGetHissLnkTxCounts: Accumulating Counts for Received Cells .76 4.11.2 duplexGetHissLnkTxCounts: Accumulating Counts for All Cells .77 4.11.3 duplexGetStatisticCounts: Recrumulating Counts for All Cells .77 4.11.4 duplexGetStatisticCounts: Resetting Driver Statistical Counts .79 4.12.1 indication Callbacks .80 4.12.1 indDuplexNotify: Notifying the Application of Significant Events .80 4.12.1 indDuplexRXEOC: Notifying the Application of Ready Extract-Cell-FIFOs .81 5 Real-Time-OS Interface Functions .83 5.1 Memory Allocation and Deallocation .84 5.1.1 sysDuplexMemFree: Deallocating Memory .84 5.1.2 sysDuplexMemFree: Deallocating Memory .86 5.2.2 duplexGetIndBuf: Getting DPR Buffers .86 5.3.1 sysDuplexDelayFn: Delaying Functions .86 5.4.1 sysDuplexDelayFn: Delaying Functions .86 5.3.1 sysDuplexSemTake: Taking Semaphores .87 5.4.2 sysDuplexSemTake: Taking Semaphores .87 5.4.3 sysDuplexSemTake: Taking Semaphores .87		4.11	Statistics Collection	76	
4.11.2 duplexGetHists.InkTxCounts: Accumulating Counts for Tiansmitted Cells 77 4.11.3 duplexGetAlHissCounts: Retrieving Driver Statistical Counts 78 4.11.4 duplexResetStatisticCounts: Retrieving Driver Statistical Counts 79 4.11.5 duplexResetStatisticCounts: Resetting Driver Statistical Counts 79 4.12 Indication Callbacks 80 4.12.1 indDuplexRbDC: Notifying the Application of Significant Events 80 4.12.3 indDuplexRbC: Notifying the Application of Received BOC 80 4.12.3 indDuplexRcell: Notifying the Application of Received BOC 80 4.12.3 indDuplexRcell: Notifying the Application of Ready Extract-Cell-FIFOs 81 5 Real-Time-OS Interface Functions 83 5.1 sysDuplexMemAlic: Allocating Memory 84 5.1.1 sysDuplexMemAlic: Allocating Memory 85 5.2 duplexGetIndBut: Getting DPR Buffers 86 5.2.1 duplexGetIndBut: Getting DPR Buffers 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.4.1 sysDuplexSemDiete: Creating Semaphores 87 5.4.2 sysDuplexSemErate: Creating Semaphores			4.11.1 duplexGetHssLnkRxCounts: Accumulating Counts for Received Cells	76	
4.11.3 duplexGetAlHsSCounts: Retriving Driver Statistical Counts 77 4.11.5 duplexResetStatisticCounts: Retriving Driver Statistical Counts 79 4.12 IndDuplexNetDity: Notifying the Application of Significant Events 80 4.12.1 indDuplexRxBCC: Notifying the Application of Received BOC 80 4.12.3 indDuplexRxBCC: Notifying the Application of Received BOC 80 4.12.3 indDuplexRxBCC: Notifying the Application of Received BOC 80 4.12.3 indDuplexRxCell: Notifying the Application of Received BOC 80 5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemFree: Deallocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexReturnIndBuf: Returning DPR Buffers 86 5.3 Timer Operations 86 5.4 sysDuplexSemTrate: Taking Semaphores 87 5.4.2 sysDuplexSemTake: Taking Semaphores 87 5.4.3 sysDuplexSemTake: Taking Semaphores 87 6 Hardware Inte			4.11.2 duplexGetHssLnkTxCounts: Accumulating Counts for Transmitted Cells	77	
4.11.4 duplexGetStatisticCounts: 79 4.11.5 duplexResetStatisticCounts: 79 4.12 Indication Callbacks. 80 4.12.1 indDuplexNotify: Notifying the Application of Significant Events 80 4.12.2 indDuplexRxBCC: Notifying the Application of Received BOC 80 4.12.3 indDuplexRxCell: Notifying the Application of Received BOC 80 4.12.3 indDuplexRxCell: Notifying the Application of Received BOC 80 5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemAlloc: Allocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 86 5.3.1 sysDuplexMemFree: Deallocating Semaphores 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.3 sysDuplexSemDelete: Deleting Semaphores 87 5.4.4 sysDuplexSemGreate: Creating Semaphores 87 5.4.3 sysDuplexSemGreate: Creating Taking Semaphores 87 5.4.4 sysDuplexSemGreate: Creating Taking Semaphores 87 6.1 Device Register Access 89			4.11.3 duplexGetAllHssCounts: Accumulating Counts for All Cells	78	
4.11.5 duplexResetStatisticCounts: Resetting Driver Statistical Counts. 79 4.12 Indication Callbacks. 80 4.12.1 indDuplexNotify: Notifying the Application of Significant Events 80 4.12.3 indDuplexRXBCC: Notifying the Application of Received BOC 80 4.12.3 indDuplexRXCell: Notifying the Application of Received BOC 80 5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemFree: Deallocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 86 5.3 Timer Operations 86 5.3.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemTake: Taking Semaphores 87 5.4.3 sysDuplexSemTake: Taking Semaphores 87 5.4.4 sysDuplexSemTake: Taking Semaphores 87 5.4.3 sysDuplexSemTake: Taking Semaphores 87 5.4.4 sysDuplexRawRead: Reading from Register Address Locations 90			4.11.4 duplexGetStatisticCounts: Retrieving Driver Statistical Counts	79	
4.12 Indication Callbacks 80 4.12.1 indDuplexRXBQC: Notifying the Application of Received BOC 80 4.12.2 indDuplexRXDCI: Notifying the Application of Received BOC 80 4.12.3 indDuplexRXCEII: Notifying the Application of Ready Extract-Cell-FIFOs 81 5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemAlloc: Allocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 86 5.2.2 duplexGetIndBuf: Returning DPR Buffers 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.3 sysDuplexSemCreate: Creating Semaphores 87 5.4.4 sysDuplexSemCreate: Creating Semaphores 87 5.4.4 sysDuplexSemGreate: Boleting Semaphores 87 6.1 Device Register Access 89 6.1.1 sysDuplexRemWreis Writing to Register Address Locations 90 6.1.2 </td <td></td> <td></td> <td>4.11.5 duplexResetStatisticCounts: Resetting Driver Statistical Counts</td> <td>79</td>			4.11.5 duplexResetStatisticCounts: Resetting Driver Statistical Counts	79	
4.12.1 indDuplexNotify: Notifying the Application of Significant Events. 80 4.12.2 indDuplexRxBDC: Notifying the Application of Received BOC. 80 4.12.3 indDuplexRxCell: Notifying the Application of Received BOC. 80 4.12.3 indDuplexRxCell: Notifying the Application of Received BOC. 80 4.12.3 indDuplexRxCell: Notifying the Application of Ready Extract-Cell-FIFOs 81 5 Real-Time-OS Interface Functions. 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemFree: Deallocating Memory 84 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 85 5.2.2 duplexGetIndBuf: Returning DPR Buffers 86 5.3 Timer Operations 86 5.4.1 sysDuplexDelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4.2 sysDuplexSemTeate: Creating Semaphores 87 5.4.2 sysDuplexSemGive: Giving Semaphores 87 5.4.3 sysDuplexSemGive: Giving Semaphores 87 6 Hardware Interface Functions 89 <td< td=""><td></td><td>4 4 9</td><td>Indiantian Callbacka</td><td>80</td></td<>		4 4 9	Indiantian Callbacka	80	
4.12.1 indDuplexRxBCC: Notifying the Application of Simical Events 30 4.12.3 indDuplexRxDCell: Notifying the Application of Received BOC 80 5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemAlloc: Allocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.3.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemCreate: Creating Semaphores 87 5.4.3 sysDuplexSemGreate: Creating Semaphores 87 5.4.3 sysDuplexSemGreate: Creating Semaphores 87 5.4.4 sysDuplexSemGreate: Creating Semaphores 87 5.4.4 sysDuplexRemGreate: Reading from Register Address Locations 90 6.1.1 sysDuplexRemWread: Reading from Register Address Locations 90 6.1.2 sysDuplexRemWread: Reading from Register Address Locations 90 6.1.1 sysDuplexRemWread: Reading		4.12	100Callon Callbacks	00	
4.12.3 indDuplexRxCell: Notifying the Application of Ready Extract-Cell-FIFOs			4.12.1 IndDuplexNotity. Notifying the Application of Significant Events	00	
4.12.3 INDUDIEXACCEI. NUMPING THE Application of Ready EXTRECE CERPTICOS 61 5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemAlloc: Allocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 84 5.2.1 duplexGetIndBuf: Getting DPR Buffers 85 5.2.2 duplexReturnIndBuf: Returning DPR Buffers 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemDeletic: Deleting Semaphores 87 5.4.3 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 6.1 Device Register Access 89 6.1.1 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexInternupt Statuses 91 6.2.1 duplexISR: Registering Interrupt Service Functions 93			4.12.2 IndDuplexRXBOC. Notifying the Application of Received BOC	00	
5 Real-Time-OS Interface Functions 83 5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemAlloc: Allocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 85 5.2.2 duplexReturnIndBuf: Returning DPR Buffers 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemDelete: Delating Semaphores 87 5.4.3 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 6.1 Device Register Access 89 6.1 Device Register Access 89 6.1.1 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWrite: Writing to Register Address Locations 90 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupt Service F			4.12.3 IndDuplexRxCell. Notifying the Application of Ready Extract-Cell-FIFOS	01	
5.1 Memory Allocation and Deallocation 84 5.1.1 sysDuplexMemAlloc: Allocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 85 5.2.2 duplexReturnIndBuf: Returning DPR Buffers 86 5.3 Timer Operations 86 5.4 sysDuplexDelayFn: Delaying Functions 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemCreate: Creating Semaphores 87 5.4.3 sysDuplexSemTake: Taking Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexRawRead: Reading from Register Address Locations 90 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.3 sysDuplexRawRead: Reading Interrupt Address Locations 90 6.1.1 sysDuplexRawRead: Reading Interrupt Service Functions 92 6.2.1 duplexDRP: Processing Interrupt Service Functions 93 6	5	Real-	Time-OS Interface Functions	83	
5.1.1 sysDuplexMemAlloc: Allocating Memory 84 5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexReturnIndBuf: Getting DPR Buffers 86 5.3 Timer Operations 86 5.4 sysDuplexNelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemCreate: Creating Semaphores 87 5.4.3 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.3 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexNthite: Writing to Register Address Locations 90 6.2		5.1	Memory Allocation and Deallocation		
5.1.2 sysDuplexMemFree: Deallocating Memory 85 5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 86 5.2.2 duplexReturnIndBuf: Returning DPR Buffers 86 5.3 Timer Operations 86 5.3.1 sysDuplexDelayFn: Delaying Functions 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemCreate: Creating Semaphores 87 5.4.3 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexRemGive: Giving Semaphores 87 6.1 Device Register Access 89 6.1 Device Register Access 89 6.1.1 sysDuplexAmwRead: Reading from Register Address Locations 90 6.1.2 sysDuplexNer Processing Interrupt Statuses 92 6.2.2 duplexIDPR: Processing Interrupt Statuses 92 6.2.3 sysDuplexIntheatulHandler: Installing Interrupt Service Functions 94 6.2.5 sysDuplexInthteatulHandler: Calling d		0	5.1.1 svsDuplexMemAlloc: Allocating Memory		
5.2 Buffer Management			5.1.2 svsDuplexMemFree: Deallocating Memory		
5.2 Buffer Management 85 5.2.1 duplexGetIndBuf: Getting DPR Buffers 85 5.2.2 duplexReturnIndBuf: Returning DPR Buffers 86 5.3 Timer Operations 86 5.3 Timer Operations 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemDelete: Deleting Semaphores 87 5.4.3 sysDuplexSemTake: Taking Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexRawRead: Reading from Register Address Locations 90 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawRead: Reading Interrupt Service Sections 90 6.2 Interrupt Servicing 91 6.2.1 duplexDR: Processing Interrupt Statuses 92 6.2.3 sysDuplexIntRemoveHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntR					
5.2.1 duplexGetIndBuf: Getting DPR Buffers .85 5.2.2 duplexReturnIndBuf: Returning DPR Buffers .86 5.3 Timer Operations .86 5.3 Timer Operations .86 5.4 Semaphore Operations .86 5.4.1 sysDuplexSemCreate: Creating Semaphores .87 5.4.2 sysDuplexSemDelete: Deleting Semaphores .87 5.4.3 sysDuplexSemTake: Taking Semaphores .87 5.4.4 sysDuplexSemGive: Giving Semaphores .87 5.4.4 sysDuplexRawRead: Reading from Register Address Locations .90 6.1 Device Register Access .89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations .90 6.1.3 sysDuplexRawRead: Reading from Register Address Locations .90 6.2.1 duplexISR: Registering Interrupt Statuses .92 6.2.2 duplexIDPR: Processing Interrupt Service Functions .93 6.2.4 sysDuplexIntHandler: Installing Interrupt Service Functions .93 6.2.5 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions .94 6.2.6 sysDuplexIntPRTask: Calling duplexISR		5.2	Buffer Management	85	
5.2.2 duplexReturnIndBuf: Returning DPR Buffers .86 5.3 Timer Operations .86 5.3.1 sysDuplexDelayFn: Delaying Functions .86 5.4 Semaphore Operations .86 5.4.1 sysDuplexSemCreate: Creating Semaphores .87 5.4.2 sysDuplexSemTake: Taking Semaphores .87 5.4.3 sysDuplexSemGive: Giving Semaphores .87 5.4.4 sysDuplexSemGive: Giving Semaphores .87 6 Hardware Interface Functions .89 6.1 Device Register Access .89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations .90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations .90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses .90 6.2 Interrupt Servicing .91 6.2.1 duplexDRR: Processing Interrupt Satuses .92 6.2.3 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions .93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions .94 6.2.5 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions			5.2.1 duplexGetIndBuf: Getting DPR Buffers	85	
5.3 Timer Operations			5.2.2 duplexReturnIndBuf: Returning DPR Buffers	86	
5.3 Timer Operations 86 5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemTake: Taking Semaphores 87 5.4.3 sysDuplexSemTake: Taking Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 6 Hardware Interface Functions 89 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDR: Processing Interrupt Service Functions 93 6.2.4 sysDuplexInthstallHandler: Installing Interrupt Service Functions 93 6.2.5 sysDuplexInthermoveHandler: Removing Interrupt Service Functions 94 6.2.6 sysDuplexDPRTask: Calling duplexISR 94		53	Timer Operations	86	
5.3.1 sysbuplexbelay in Belaying Functions		5.5	5.3.1 sysDupleyDelayEn: Delaying Functions	86	
5.4 Semaphore Operations 86 5.4.1 sysDuplexSemCreate: Creating Semaphores 87 5.4.2 sysDuplexSemTake: Taking Semaphores 87 5.4.3 sysDuplexSemGive: Giving Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 6 Hardware Interface Functions 89 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWrite: Writing to Register Address Locations 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupt Service Functions 93 6.2.4 sysDuplexInthstallHandler: Installing Interrupt Service Functions 94 6.2.5 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.6 sysDuplexIntr			3.3.1 SysDuplexDelayi II. Delaying I unctions	00	
5.4.1 sysDuplexSemCreate: Creating Semaphores. 87 5.4.2 sysDuplexSemTake: Taking Semaphores. 87 5.4.3 sysDuplexSemGive: Giving Semaphores. 87 5.4.4 sysDuplexSemGive: Giving Semaphores. 87 6 Hardware Interface Functions 89 6.1 Device Register Access. 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRead: Reading from Register Address Locations 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupts 92 6.2.3 sysDuplexInthstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntRadler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97		5.4	Semaphore Operations	86	
5.4.2 sysDuplexSemDelete: Deleting Semaphores. 87 5.4.3 sysDuplexSemTake: Taking Semaphores. 87 5.4.4 sysDuplexSemGive: Giving Semaphores. 87 6 Hardware Interface Functions 89 6.1 Device Register Access. 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWirite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWirite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWirite: Writing to Register Address Locations 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 93 6.2.5 sysDuplexIntRemoveHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 <			5.4.1 sysDuplexSemCreate: Creating Semaphores	87	
5.4.3 sysDuplexSemTake: Taking Semaphores 87 5.4.4 sysDuplexSemGive: Giving Semaphores 87 6 Hardware Interface Functions 89 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.3 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupt Statuses 92 6.2.3 sysDuplexInthstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexDPRTask: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98			5.4.2 sysDuplexSemDelete: Deleting Semaphores	87	
5.4.4 sysDuplexSemGive: Giving Semaphores. 87 6 Hardware Interface Functions 89 6.1 Device Register Access. 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexRawWrite: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDPR: Processing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Installing Interrupt Service Functions 93 6.2.5 sysDuplexIntRemoveHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98			5.4.3 sysDuplexSemTake: Taking Semaphores	87	
6 Hardware Interface Functions 89 6.1 Device Register Access 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupt Service Functions 93 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Porting Procedures 97 7.2 Driver Porting Driver OS Extensions 98			5.4.4 sysDuplexSemGive: Giving Semaphores	87	
6.1 Device Register Access. 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98	6	Hard	vare Interface Eurotions	80	
6.1 Device Register Access. 89 6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98	0	Hardware Interface Functions			
6.1.1 sysDuplexRawRead: Reading from Register Address Locations 90 6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexISR: Registering Interrupt Statuses 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98		6.1	Device Register Access	89	
6.1.2 sysDuplexRawWrite: Writing to Register Address Locations 90 6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses 90 6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexIPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98			6.1.1 svsDuplexRawRead: Reading from Register Address Locations	90	
6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses .90 6.2 Interrupt Servicing .91 6.2.1 duplexISR: Registering Interrupt Statuses .92 6.2.2 duplexDPR: Processing Interrupts .92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions .92 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions .93 6.2.5 sysDuplexIntHandler: Calling duplexISR .94 6.2.6 sysDuplexDPRTask: Calling duplexDPR .95 7 Porting Drivers .97 7.1 Driver Source Files .97 7.2 Driver Porting Procedures .97 7.2.1 Porting Driver OS Extensions .98			6.1.2 svsDuplexRawWrite: Writing to Register Address Locations	90	
6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98			6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses	90	
6.2 Interrupt Servicing 91 6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98					
6.2.1 duplexISR: Registering Interrupt Statuses 92 6.2.2 duplexDPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98		6.2	Interrupt Servicing	91	
6.2.2 duplexDPR: Processing Interrupts 92 6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexIntHandler: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98			6.2.1 duplexISR: Registering Interrupt Statuses	92	
6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions 93 6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions 94 6.2.5 sysDuplexIntHandler: Calling duplexISR 94 6.2.6 sysDuplexDPRTask: Calling duplexDPR 95 7 Porting Drivers 97 7.1 Driver Source Files 97 7.2 Driver Porting Procedures 97 7.2.1 Porting Driver OS Extensions 98			6.2.2 duplexDPR: Processing Interrupts	92	
6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions			6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions	93	
6.2.5 sysDuplexIntHandler: Calling duplexISR			6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions	94	
6.2.6 sysDuplexDPRTask: Calling duplexDPR			6.2.5 sysDuplexIntHandler: Calling duplexISR	94	
7 Porting Drivers			6.2.6 sysDuplexDPRTask: Calling duplexDPR	95	
 7.1 Driver Source Files	7	Portir	ng Drivers	97	
 7.1 Driver Source Files		71	Driver Source Files	07	
7.2 Driver Porting Procedures		1.1		97	
7.2.1 Porting Driver OS Extensions		7.2	2 Driver Porting Procedures		
			7.2.1 Porting Driver OS Extensions	98	

PMC-990799

DRIVER MANUAL



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

	7.2.2 7.2.3 7.2.4	Porting Drivers to Hardware Platforms Porting Driver Application-Specific Elements Building Drivers	
Appendix	: Coding	g Conventions	
Acronym	s		
Index			
Contactir	ng PMC-	Sierra, Inc	



S/UNI-DUPLEX DRIVER MANUAL

LIST OF FIGURES

Figure 1: Driver Architecture
Figure 2: Driver Software States
Figure 3: Device Initialization, Re-initialization, and Shutdown22
Figure 4: Cell Extraction23
Figure 5: Interrupt Service Model24
Figure 6: Polling Service Model
Figure 7: Application Interface44
Figure 8: Real-Time OS Interface
Figure 9: Hardware Interface
Figure 10: Driver Source Files

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL



S/UNI-DUPLEX DRIVER MANUAL

LIST OF TABLES

Table 1: Driver Functions and Features 15
Table 2: Driver Software States
Table 3: Cell Header Structure: sDPX_CELL_HDR 29
Table 4: Cell Control Structure: sDPX_CELL_CTRL
Table 5: Initialization Vector: sDPX_INIT_VECTOR
Table 6: Device Registers: sDPX_REGS
Table 7: SCI-PHY/Any-PHY Registers: sDPX_SCI_ANY_PHY_REGS
Table 8: HSS Link Registers: sDPX_HSS_REGS32
Table 9: Clocked Serial-Interface Registers: sDPX_CLK_SER_REGS
Table 10: Global Driver Database: sDPX_GDD
Table 11: Device Data Block: sDPX_DDB34
Table 12: Interrupt Enables: sDPX_INT_ENBLS
Table 13: Interrupt Context: sDPX_INT_CTXT
Table 14: HSS Counts: sDPX_HSS_CNTS
Table 15: Statistical Counts: sDPX_STAT_COUNTS
Table 16: Definition of Variable Types 105
Table 17: Variable Naming Conventions106
Table 18: Function and Macro Naming Conventions 107

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

1 DRIVER PORTING QUICK START

This section summarizes how to port the S/UNI-DUPLEX device driver to your hardware and operating system (OS) platform.

Note: Because each platform and application is unique, this manual can only offer guidelines for porting the S/UNI-DUPLEX driver.

The code for the S/UNI-DUPLEX driver is organized into C source files. You may need to modify the code or develop additional code. The code is in the form of constants, macros, and functions. For the ease of porting, the code is grouped into source files (src) and include files (inc). The src files contain the functions and the inc files contain the constants and macros.

To port the S/UNI-DUPLEX driver to your platform:

- 1. Port the driver's OS extensions (page 98):
 - Data types
 - OS-specific services
 - Utilities and interrupt services that use OS-specific services
- 2. Port the driver to your hardware platform (page 100):
 - Port the device detection function.
 - Port low-level device read-and-write macros.
 - Define hardware system-configuration constants.
- 3. Port the driver's application-specific elements (page 102):
 - Define the task-related constants.
 - Code the callback functions.
- 4. Build the driver (page 103).

For more information about porting the S/UNI-DUPLEX driver, see section 7.

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

2 DRIVER FUNCTIONS AND FEATURES

The following table lists the main functions and features offered by the S/UNI-DUPLEX driver. You can alter these functions by modifying or adding to the driver's code.

Table 1: Driver	Functions	and Features
-----------------	-----------	--------------

Functions	Description
Device Addition and Deletion (page 45)	 These functions perform the following tasks: Reset new devices Allocate and initialize memory that will store context information for new devices Deallocate device context memory during device shutdown
Device Initialization (page 49)	These functions initialize the S/UNI-DUPLEX device and its associated context structures.
Device Diagnostics (page 53)	These functions write values to registers and read them back to verify the microprocessor's input and output interface with the device. They enable and disable internal and external loopback for the S/UNI-DUPLEX device's high-speed serial (HSS) interfaces. They also monitor the device's clocks.
HSS Link Configuration (page 56)	These functions configure the HSS links of the S/UNI-DUPLEX device by programming the HSS link registers according to the parameters specified.
Cell Insertion and Extraction (page 59)	These functions insert cells into, and extract cells from, the S/UNI-DUPLEX device control channels by manipulating the insert and extract FIFO control and status registers.
BOC Transmission and Reception (page 65)	These functions transmit and receive BOC on the HSS interfaces. Writing to the transmit-BOC registers transmits BOC. BOC is received by monitoring the receive-BOC status-registers.

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Statistics Collection (page 67)	These functions retrieve the device counts (including cells received, cells transmitted, and errored cells received) for accumulation by the application.
Interrupt Servicing (page 23)	These functions clear the interrupts raised by the S/UNI-DUPLEX device. Then they queue the interrupt status for later processing by a deferred interrupt-processing routine (DPR). The DPR runs in the context of a separate task within the RTOS and takes appropriate actions based on the interrupt status queued for it by the Interrupt Servicing Routine (ISR). In polling mode, the DPR process periodically services the interrupt status.
Indication Callbacks (page 80)	The DPR uses indication callback functions to notify the application of events in the S/UNI-DUPLEX device and driver. These events include the reception of cells in the microprocessor extract cell FIFOs and the reception of valid BOC.

2.1 Driver Architecture

The driver includes seven main modules:

- Driver API module
- Real-time-OS interface module
- Hardware interface module
- Driver library module
- Device data-block module
- Interrupt-service routine module
- Deferred-processing routine module

For more information about these modules, see the following sections.

Figure 1 illustrates the architectural modules of the S/UNI-DUPLEX driver.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Figure 1: Driver Architecture



2.1.1 Driver API Module

The driver's API is a collection of high level functions that can be called by application programmers to configure, control, and monitor the S/UNI-DUPLEX device, such as:

- Initializing the device
- Validating device configuration
- Retrieving device status and statistics information.
- Diagnosing the device

The driver API functions use the driver library functions as building blocks to provide this system level functionality to the application programmer (see below).





S/UNI-DUPLEX DRIVER MANUAL

The driver API also consists of callback functions that notify the application of significant events that take place within the device and driver, including cell and BOC reception.

2.1.2 Driver Real-Time-OS Interface Module

The driver's RTOS interface module provides functions that let the driver use RTOS services. The S/UNI-DUPLEX driver requires the memory, interrupt, and preemption services from the RTOS. The RTOS interface functions perform the following tasks for the S/UNI-DUPLEX device and driver:

- Allocate and deallocate memory
- Manage buffers for the DPR
- Pause task execution
- Manage semaphores

The RTOS interface also includes service callbacks. These are functions installed by the driver using RTOS service calls, such as install interrupts and start timers. These service callbacks are invoked when an interrupt occurs or a timer expires.

Note: You must modify RTOS interface code to suit your RTOS.

2.1.3 Driver Hardware-Interface Module

The S/UNI-DUPLEX hardware interface provides functions that read from and write to S/UNI-DUPLEX device-registers. The hardware interface also provides a template for an ISR that the driver calls when the device raises a hardware interrupt. You must modify this function based on the interrupt configuration of your system.

2.1.4 Driver Library Module

The driver library module is a collection of low-level utility functions that manipulate the device registers and the contents of the driver's DDB. The driver library functions serve as building blocks for higher level functions that constitute the driver API module. Application software does not normally call the driver library functions.

2.1.5 Device Data-Block Module

The DDB stores context information about the S/UNI-DUPLEX device, such as:

PRELIMINARY DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

- Device state
- Control information
- Initialization vector
- Callback function pointers
- Statistical counts

The driver allocates context memory for the DDB when the driver registers a new device.

2.1.6 Interrupt-Service Routine Module

The S/UNI-DUPLEX driver provides an ISR called duplexISR that checks if there are any valid interrupt conditions present for the device. This function can be used by a system-specific interrupt-handler function to service interrupts raised by the device.

The low-level interrupt-handler function that traps the hardware interrupt and calls duplexISR is system and RTOS dependent. Therefore, it is outside the scope of the driver. An example implementation of such an interrupt handler (see page 94) as well as installation and removal functions (see page 93 and page 94) is provided as a reference. You can customize these example implementations to suit your specific needs.

See page 23 for a detailed explanation of the ISR and interrupt-servicing model.

2.1.7 Deferred-Processing Routine Module

The DPR provided by the S/UNI-DUPLEX driver (duplexDPR) clears and processes interrupt conditions for the device. Typically, a system specific function, which runs as a separate task within the RTOS, executes the DPR.

See page 23 for a detailed explanation of the DPR and interrupt-servicing model.

2.2 Driver Software States

Figure 2 shows the software state diagram for the S/UNI-DUPLEX driver. State transitions occur on the successful execution of the corresponding transition functions shown. State information helps maintain the integrity of the driver's DDB by controlling the set of device operations allowed in each state. Table 2 describes the software states for the S/UNI-DUPLEX device as maintained by the driver.

DRIVER MANUAL PMC-990799 PMC PMC-Sierra, Inc.

PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL





Table 2: Driver Software States

State	Description
Empty	The S/UNI-DUPLEX device is not registered. This is the initial state.
Present	The driver has detected the S/UNI-DUPLEX device and the drive has passed power-on self-tests. The driver has allocated memory to store context information about this device.
Init	An initialization vector passed by the application has successfully initialized the S/UNI-DUPLEX device. The initialization parameters have been validated and the device has been configured by writing appropriate bits in the control registers of the device.
Active	The S/UNI-DUPLEX device has been activated. This means that the device interrupts have been enabled and the device is ready for normal operation.

2.3 Processing Flows

This section describes some of the main processing flows of the S/UNI-DUPLEX driver:

• Device initialization, re-initialization, and shutdown





S/UNI-DUPLEX DRIVER MANUAL

- Cell extraction
- Interrupt servicing
- Polling servicing

The flow diagrams presented here illustrate the sequence of operations that take place for different driver functions. The diagrams also serve as a guide to the application programmer by illustrating the sequence in which the application must invoke the driver API.

2.3.1 Device Initialization, Re-initialization, and Shutdown

The following figure shows the functions and process that the driver uses to initialize, re-initialize, and shutdown the S/UNI-DUPLEX device.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Figure 3: Device Initialization, Re-initialization, and Shutdown



2.3.2 Cell Extraction

The following figure shows the functions and process that the driver uses to extract cells from the S/UNI-DUPLEX device.

DRIVER MANUAL PMC-990799

ISSUE 1

PM7350 S/UNI-DUPLEX DRIVER

S/UNI-DUPLEX DRIVER MANUAL

Figure 4: Cell Extraction



The deferred processing routine invokes this indication callback function to inform the application of a cell reception. The indDuplexRxCell function is typically implemented as a message queuing function that sends a message to another task (referred to henceforth as the cell reception task) that is dedicated to process received cells. The deferred processing routine also disables further RX indications.

PMC-Sierra, Inc.

The cell reception task now checks the status of the Extract FIFOs of the S/UNI-DUPLEX device. This function determines which extract FIFOs have cells to be dequeued.

Cells are now dequeued by repeatedly invoking duplexExtractCell till the Extract FIFOs are empty. The message completion is detected by an End of Message bit in a cell type flag output from pCellTypeFn function. The function is installed by the application as a callback function. The Extract FIFOs are again checked to see if there are any more cells to be extracted.

After extracting all the cells from the Extract FIFOs of the S/UNI-DUPLEX device, the cell reception task re-enables the RX indication for the device.

2.3.3 Interrupt Servicing

The S/UNI-DUPLEX driver services device interrupts using an interrupt service routine (ISR) that traps interrupts and a deferred interrupt-processing routine (DPR) that actually processes the interrupt conditions and clears them. This lets the ISR execute quickly and exit. Most of the time-consuming processing of the interrupt conditions is deferred to the DPR by queuing the necessary interrupt-context information to the DPR task. The DPR function runs in the context of a separate task within the RTOS.

Note: Since the DPR task processes potentially serious interrupt conditions, you should set the DPR task's priority higher than the application task interacting with the S/UNI-DUPLEX driver.

The driver provides system-independent functions, duplexISR and duplexDPR. You must fill in the corresponding system-specific functions, sysDuplexISR and sysDuplexDPR. The system-specific functions isolate the system-specific communication mechanism (between the ISR and DPR) from the system-independent functions, duplexISR and duplexDPR.

Figure 5 illustrates the interrupt service model used in the S/UNI-DUPLEX driver design.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Figure 5: Interrupt Service Model



Note: Instead of using an interrupt service model, you can use a polling service model in the S/UNI-DUPLEX driver to process the device's event-indication registers (see page 26).

Calling duplexISR

An interrupt handler function, which is system dependent, must call duplexISR. But first, the low-level interrupt-handler function must trap the device interrupts. You must implement this function for your system. As a reference, an example implementation of the interrupt handler (sysDuplexIntHandler) appears on page 94. You can customize this example implementation to suit your needs.

The interrupt handler that you implement (sysDuplexIntHandler) is installed in the interrupt vector table of the system processor. Then it is called when one or more S/UNI-DUPLEX devices interrupt the processor. The interrupt handler then calls duplexISR for each device in the active state.

The duplexISR function reads from the master interrupt-status register and the miscellaneous interrupt-status register of the S/UNI-DUPLEX. Then duplexISR returns with this status information if a valid status bit is set. If a valid status bit is set, the duplexISR also disables that device's interrupts. The sysDuplexIntHandler function then sends a message to the DPR task that consists of the device handles of all the S/UNI-DUPLEX devices that had valid interrupt conditions.

Note: Normally you should save the status information for deferred interrupt processing by implementing a message queue. The interrupt handler sends the status information to the queue by the sysDuplexIntHandler.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Calling duplexDPR

The sysDuplexDPRTask function is a system specific function that runs as a separate task within the RTOS. You should set the DPR task's priority higher than the application task(s) interacting with the S/UNI-DUPLEX driver. In the message-queue implementation model, this task has an associated message queue. The task waits for messages from the ISR on this message queue. When a message arrives, sysDuplexDPRTask calls the DPR (duplexDPR).

Then duplexDPR processes the status information and takes appropriate action based on the specific interrupt condition detected. The nature of this processing can differ from system to system. Therefore, duplexDPR calls different indication callbacks for different interrupt conditions.

Typically, you should implement these callback functions as simple message posting functions that post messages to an application task. However, you can implement the indication callback to perform processing within the DPR task context and return without sending any messages. In this case, ensure that the indication function does not call any API functions that change the driver's state, such as duplexDelete. Also, ensure that the indication function function is non-blocking because the DPR task executes while S/UNI-DUPLEX interrupts are disabled. You can customize these callbacks to suit your system. See page 80 for a description of the callback functions.

Note: Since the duplexISR and duplexDPR routines themselves do not specify a communication mechanism, you have full flexibility in choosing a communication mechanism between the two. A convenient way to implement this communication mechanism is to use a message queue, which is a service that most RTOSs provide.

You must implement the two system specific functions, sysDuplexIntHandler and sysDuplexDPRTask. When the driver calls sysDuplexIntInstallHandler for the first time, the driver installs sysDuplexIntHandler in the interrupt vector table of the processor. The sysDuplexDPRTask function is also spawned as a task during this first time invocation of sysDuplexIntInstallHandler. The sysDuplexIntInstallHandler function also creates the communication channel between sysDuplexIntHandler and sysDuplexDPRTask. This communication channel is most commonly a message queue associated with the sysDuplexDPRTask.

Similarly, during removal of interrupts, the driver removes sysDuplexIntHandler from the microprocessor's interrupt vector table and deletes the task associated with sysDuplexDPRTask.





S/UNI-DUPLEX DRIVER MANUAL

As a reference, this manual provides example implementations of the interrupt installation and removal functions. For more information about the interrupt installation function and prototype, see page 93. For more information about the interrupt removal function and prototype, see page 94. You can customize these prototypes to suit your specific needs.

2.3.4 Polling Servicing

Instead of using an interrupt service model, you can use a polling service model in the S/UNI-DUPLEX driver to process the device's event-indication registers.

Figure 6 illustrates the polling service model used in the S/UNI-DUPLEX driver design.

Figure 6: Polling Service Model



The polling service code includes some system specific code (prefixed by "sysDuplex"), which typically you must implement for your application. The polling service code also includes some system independent code (prefixed by "duplex") provided by the driver that does not change from system to system.

In polling mode, sysDuplexIntHandler and duplexISR are not used. Instead, the driver spawns a sysDuplexDPRTask function as a task processor when the driver calls sysDuplexIntInstallHandler for the first time.

In sysDuplexDPRTask, the driver-supplied DPR (duplexDPR) is periodically called for each device in the active state. The duplexDPR reads from the master interrupt-status and miscellaneous interrupt-status registers of the S/UNI-DUPLEX. If some valid status bits are set, it processes the status information and takes appropriate action based on the specific interrupt condition detected.





S/UNI-DUPLEX DRIVER MANUAL

The nature of this processing can differ from system to system. Therefore, the DPR calls different indication callbacks for different interrupt conditions. You can customize these callbacks to fit your application's specific requirements. See page 80 for a description of these callback functions.

Similarly, during removal of polling service, the driver removes the task associated with sysDuplexDPRTask if none of S/UNI-DUPLEX devices is activated.

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

3 DRIVER DATA STRUCTURES

The S/UNI-DUPLEX driver uses several data structures. These structures help to:

- Control and store cell header information
- Configure the S/UNI-DUPLEX device
- Identify the device's context
- Support interrupt processing
- Store indication callbacks

3.1 Cell Data Structures

This section describes the data structures that the driver uses to help control cell insertion and extraction. These structures serve as templates for received and transmitted cells.

3.1.1 Cell-Header Data Structure

The following structure stores cell header data.

Table 3: Cell Header Structure: sDPX_CELL_HDR

Member Name	Туре	Description
ulUsrPrpnd[2]	UINT1	Two prepend bytes that you specify
ulHdr[5]	UINT1	H1-H5 cell header bytes
ulUDF	UINT1	A field you define

3.1.2 Cell-Control Data Structure

The following structure controls cell insertion and extraction operations.

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Table 4: Cell Control Structure: sDPX_CELL_CTRL

Member Name	Туре	Description	
u4Crc32Prev	UINT4	The CRC-32 value in the insert and extract CRC-32 accumulator registers after the previous cell was inserted or extracted. Used to preset the accumulator registers before inserting or extracting the next cell.	
u4Crc32	UINT4	The CRC-32 value in the insert and extract accumulator registers after the current cell was inserted or extracted.	
ulCellType	UINT1	 A flag used by the driver to indicate that the cell extracted is the last cell or first cell of a message, and is CRC protected or not. Bit 0: If 1, then CRC-32 on If 0, then CRC-32 off Bit 1: If 1, then first cell If 0, then not first cell Bit 2: If 1, then last cell If 0, then not last cell 	

3.2 Device-Configuration Data Structures

This section describes the data structures that the driver uses to initialize and configure the S/UNI-DUPLEX device.

3.2.1 Initialization Data Structure

The device initialization function initializes the S/UNI-DUPLEX device and its associated context structures. This involves reading an initialization vector. The driver validates this vector and then configures the S/UNI-DUPLEX device accordingly.

The application sets the initialization vector before initializing a S/UNI-DUPLEX device. The initialization vector contains configuration parameters that the driver uses to program the S/UNI-DUPLEX device control-registers.



S/UNI-DUPLEX DRIVER MANUAL

Note: The application must free the initialization vector memory.

Table 5: Initialization Vector: sDPX_INIT_VECTOR

Member Name	Туре	Description
sRegInfo	sDPX_REGS	Contains the values that the driver will write to the control registers of the S/UNI-DUPLEX device
indNotify	DPX_IND_CB_FN	Indication callback function called by the DPR when a significant event occurs in the driver software
indRxBOC	DPX_IND_CB_FN	Indication callback function called by the DPR to forward a received valid BOC to the application
indRxCell	DPX_IND_CB_FN	Indication callback function called by the DPR when the driver must read cells from the Extract FIFOs
pCellTypeFn	DPX_CELLTYPE_FN	A Cell Type detector function that is used by the driver to determine if a cell extracted is the last or first of a particular message, and/or if it is CRC-32 protected
u4Reserved	UINT4	Placeholder for future use

3.2.2 Register Data Structure

The register data structure contains the initial values that the driver will write to the S/UNI-DUPLEX device control-registers.

Table 6: Device Registers: sDPX_REGS

Member Name	Туре	Description
ulMasterCfg	UINT1	Master configuration register



PM7350 S/UNI-DUPLEX DRIVER

DRIVER MANUAL PMC-990799

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Member Name	Туре	Description
sSciAnyPhyRegs	SDPX_SCI_ANY_PHY_REGS	SCI-PHY/Any-PHY configuration registers
sHssRegs	SDPX_HSS_REGS	HSS-link configuration registers
sClkSerRegs	SDPX_CLK_SER_REGS	Clocked-bit serial-interface configuration registers
sIntEnRegs	SDPX_INT_ENBLS	Interrupt enable registers

Table 7: SCI-PHY/Any-PHY Registers: sDPX_SCI_ANY_PHY_REGS

Member Name	Туре	Description
ulExtAddrMatch[2]	UINT1	Extended address match [2 bytes (LSB, MSB)]
ulExtAddrMask[2]	UINT1	Extended address mask [2 bytes (LSB, MSB)]
ulOutAddrMatch	UINT1	Output address match register
ulSciAnyPhyInpCfg[2]	UINT1	SCI-PHY/Any-PHY input configuration (2 bytes)
ullCAEnable[4]	UINT1	Input cell available enable (4 bytes)
ulSciAnyPhyOutCfg	UINT1	SCI-PHY/Any-PHY output configuration
ulSciAnyPhyOutPollRng	UINT1	SCI-PHY/Any-PHY output polling range

Table 8: HSS Link Registers: sDPX_HSS_REGS

Member Name	Туре	Description
ulRxCfg[2]	UINT1	Receive HSS configuration [2 bytes (RXD1, RXD2)]

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Member Name	Туре	Description
ulRxCfg[2]	UINT1	Receive HSS configuration [2 bytes (RXD1, RXD2)]
ulRxHcsPass[2]	UINT1	Receive HSS cell-filtering configuration (HCSPASS) [2 bytes (RXD1, RXD2)]
ulTxCfg	UINT1	Transmit HSS configuration

Table 9: Clocked Serial-Interface Registers: sDPX_CLK_SER_REGS

Member Name	Туре	Description
ulRxCfg[16]	UINT1	Receive serial indirect-channel configuration
ulRxLcdCntThresh[16]	UINT1	Receive serial LCD-count threshold
ulTxSerIndChnlData[16]	UINT1	Transmit serial indirect-channel data register
ulTxFrameBitThresh	UINT1	Transmit serial framing-bit threshold

3.3 **Device-Context Data Structures**

This section describes the data structures that the driver uses to store data about the S/UNI-DUPLEX device and related devices.

3.3.1 Global Driver-Database Structure

The Global Driver Database (GDD) stores module level data, such as the number of devices that the driver controls and an array of pointers to the individual device context structures (DDBs).





S/UNI-DUPLEX DRIVER MANUAL

Table 10: Global Driver Database: sDPX_GDD

Member Name	Туре	Description
ulNumDevs	UINT1	Number of devices added
pDdb[DPX_MAX_NUM_DEVS]	sDPX_DDB*	Array of pointers to the individual DDBs
u4Reserved	UINT4	Reserved for future use

3.3.2 Device Data-Block Structure

The DDB contains device context data, such as:

- Device state •
- Control data •
- Initialization vector •
- Callback function pointers •

The driver allocates the DDB memory when the driver registers a new device. The memory is deallocated when an existing device is deleted.

Table 11: Device Data Block: sDPX_DDB

Member Name	Туре	Description
usrCtxt	DPX_USR_CTXT	This variable stores the device's role in the context of your system. The driver passes it as an input parameter when the driver calls an application callback.
pSysInfo	VOID *	Pointer to system-specific device information. For example, in PCI bus environments, the bus, device, function numbers, IRQ assignment etc.

PMC-990799

DRIVER MANUAL



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Member Name	Туре	Description
u4BaseAddr	UINT4	Base address of the device
eDevMode	eDPX_MODE	<pre>Device mode, which can be one of: • DPX_SCI_MASTER • DPX_SCI_ANY_SLAVE • DPX_CLK_BIT_SER</pre>
eDevState	edpx_state	Device state, which can be one of the following enumerated type values:
		• DPX_EMPTY
		• DPX_PRESENT
		• DPX_INIT
		• DPX_ACTIVE
ulIntrProcEn	UINT1	1: Interrupt processing enabled
		0: Interrupt processing disabled
sInitVector	sDPX_INIT_VECTOR	Device configuration information passed by the application to the driver. The driver writes the appropriate S/UNI-DUPLEX device registers based on the contents of this vector.
sIntEnbls	SDPX_INT_ENBLS	Maintains a snapshot of the current interrupt-enables registers for the S/UNI-DUPLEX device
indNotify	DPX_IND_CB_FN	Indication callback function called by the DPR when a significant event occurs in the driver software

PMC-990799

DRIVER MANUAL



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Member Name	Туре	Description
indRxBOC	DPX_IND_CB_FN	Indication callback function called by the DPR to forward a received valid BOC to the application
indRxCell	DPX_IND_CB_FN	Indication callback function called by the DPR when the driver must read cells from the Extract FIFOs
pCellTypeFn	DPX_CELLTYPE_FN	A cell-type detector function that the driver uses to determine if a cell extracted is the last or first cell of a the message, and if it is CRC protected
sStatCounts	SDPX_STAT_COUNTS	Contains the statistical counts for events and the number of interrupts
lockId	DPX_SEM_ID	Semaphore ID for the data structure. It is used for mutual exclusion access to the structure.
u4Reserved	UINT4	Placeholder for future use

3.4 Interrupt Data Structures

This section describes the data structures that the S/UNI-DUPLEX driver uses to queue interrupt context data for interrupt-enable bit-setting data.

3.4.1 Interrupt-Enable Data Structure

The interrupt-enable bit-setting data is queued in the following structure.


PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Table 12: Interrupt Enables: sDPX_INT_ENBLS

Member Name	Туре	Description
ulMasterEn	UINT1	Master Interrupt Enable
ulRoolEn	UINT1	ROOLE bit (tracks change in ROOLV bit; located in clock monitor register)
ulSciAnyPhyInpIntEn	UINT1	SCI-PHY/Any-PHY Input Interrupt Enables
ulSciAnyPhyOutIntEn	UINT1	SCI-PHY/Any-PHY Output Interrupt Enable (CELLXFERRE bit)
ulMicroCellBufCtrl	UINT1	Microprocessor Cell Buffer Interrupt Control
ulRxLogChnlFifoCtrl	UINT1	Receive Logical Channel FIFO Control (FOVRE bit)
ulRxHssExtractFifoOvr[2]	UINT1	RXD1 and RXD2 Extract FIFO Control (UPF10VRE bit)
ulRxHssIntEn[2]	UINT1	Receive HSS Interrupt Enables [2 bytes (RXD1, RXD2)]
ulRxHssBocIntEn[2]	UINT1	Receive HSS BOC Interrupt Enables [2 bytes (RXD1, RXD2)]
ulTxLogChnlFifoCtrl	UINT1	Transmit Logical Channel FIFO Control (FOVRE bit)
ulRxClkSerIntEn[16]	UINT1	Receive Serial Indirect Channel Interrupt Enables

3.4.2 Interrupt-Context Data Structure

The following structure passes interrupt context data from the interrupt servicing routine to the DPR.



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Table 13: Interrupt Context: sDPX_INT_CTXT

Member Name	Туре	Description	
ulNumDevs	UINT1	Number of devices for which interrupts have to be processed	
pu4DevHandles [DPX_MAX_NUM_DEVS]	UINT4*	Array of size DPX_MAX_NUM_DEVS. The first ulNumDevs elements of this array contain the device handles for the devices for which interrupts have to be processed.	

3.5 **Count Structures**

This section describes the data structures that the S/UNI-DUPLEX driver uses to store counts.

3.5.1 HSS Counts

This section describes the data structure that the driver uses to store the number of HSS cells received and transmitted, and the number of cells that failed to be received.

Table 14: HSS Counts: sDPX HSS CNTS

Member Name	Туре	Description	
u4RxCells[2]	UINT4	Cells received count [2 words (RXD1, RXD2)]	
u4TxCells	UINT4	Cells transmitted count	
ulHcsErrs[2]	UINT1	HCS received count [2 bytes (RXD1, RXD2)]	

3.5.2 Statistical Counts

This section describes the data structure that the driver uses to store statistical counts.



C. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Table 15: Statistical Counts: sDPX_STAT_COUNTS

Member Name	Туре	Description	
Count_Tx_Hss_Count_Overflow	UINT4	Corresponds to register 0x61, bit 5	
Count_Tx_Hss_Count_Updated	UINT4	Corresponds to register 0x61, bit 6	
Count_Rx_Lc_Fifo_Overflow	UINT4	Corresponds to register 0x3D, bit 0	
Count_Tx_Lc_Fifo_Overflow	UINT4	Corresponds to register 0x5D, bit 0	
Count_Phy_Input_Cell_Xfered	UINT4	Corresponds to register 0x0F, bit 2	
Count_Invalid_SOC_Sequence	UINT4	Corresponds to register 0x0F, bit 1	
Count_Phy_Input_Parity	UINT4	Corresponds to register 0x0F, bit 0	
Count_Phy_Output_Error	UINT4	Corresponds to register 0x14, bit 7	
Count_Micro_Insert_Fifo_Ready	UINT4	Corresponds to register 0x20, bit 4	
Count_Micro_Insert_Fifo_Full	UINT4	Corresponds to register 0x20, bit 5	
Count_Extract_Cell_CRC_Error	UINT4	Corresponds to register 0x20, bit 7	
Count_Clock_Lock_Fail	UINT4	Corresponds to register 0x04, bit 3	
Count_RxSerChnl_Out_Of_Delin [16]	UINT4	Corresponds to register 0x6B, bits 0,6	
Count_RxSerChnl_In_Delin[16]	UINT4	Corresponds to register 0x6B, bits 0,6	

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

S/UNI-DUPLEX DRIVER MANUAL

Member Name	Туре	Description	
Count_RxSerChnl_Fifo_Overflow [16]	UINT4	Corresponds to register 0x6B, bit 1	
Count_RxSerChnl_HCS_Error[16]	UINT4	Corresponds to register 0x6B, bit 2	
Count_RxSerChnl_Out_Of_Sync [16]	UINT4	Corresponds to register 0x6B, bits 3,7	
Count_RxSerChnl_in_Sync[16]	UINT4	Corresponds to register 0x6B, bits 3,7	
Count_RxHss_Extract_Fifo_ Overflow[2]	UINT4	Corresponds to registers 0x31, 0x35, bit 0	
Count_RxHss_Loss_Of_Signal[2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 0 • 0x41, 0x51, bit 0	
Count_RxHss_Signal_Detected [2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 0 • 0x41, 0x51, bit 0	
Count_RxHss_Out_Of_Delin[2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 1 • 0x41, 0x51, bit 1	
Count_RxHss_In_Delin[2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 1 • 0x41, 0x51, bit 1	
Count_RxHss_Active_Bit[2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 2 • 0x41, 0x51, bit 2	
Count_RxHss_No_Active_Bit[2]	UINT4	 Corresponds to registers: 0x43, 0x53, bit 2 0x41, 0x51, bit 2 	

DRIVER MANUAL PMC-990799



Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Member Name	Туре	Description
Count_RxHss_Out_Of_Sync[2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 4 • 0x41, 0x51, bit 4
Count_RxHss_In_Sync[2]	UINT4	Corresponds to registers: • 0x43, 0x53, bit 4 • 0x41, 0x51, bit 4
Count_RxHss_CRC8_Error[2]	UINT4	Corresponds to registers 0x43, 0x53, bit 3
Count_RxHss_HCS_Error[2]	UINT4	Corresponds to registers 0x43, 0x53, bit 5
Count_RxHss_Count_Updated[2]	UINT4	Corresponds to registers 0x43, 0x53, bit 6
Count_RxHss_Count_Overflow[2]	UINT4	Corresponds to registers 0x43, 0x53, bit 7
Count_Rx_BOCs[2]	UINT4	Corresponds to registers 0x19, 0x1B, bit 6
Count_Extract_Cells	UINT4	Corresponds to register 0x20, bit 6
Count_Interrupts	UINT4	Number of interrupts occurred for the device

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4 APPLICATION INTERFACE FUNCTIONS

The driver's API is a collection of high level functions that application programmers can call to configure, control, and monitor S/UNI-DUPLEX devices.

Note: These functions are not re-entrant. This means that two application tasks cannot invoke the same API at the same time. However, the driver protects it's data structures from concurrent accesses by the application and the DPR task.

The application interface also consists of callback functions. These callback functions notify the application of significant events that take place within the device and driver, such as:

- Occurrence of critical errors
- Reception of cells
- Reception of valid BOCs

The duplexDPR routine invokes the indication callback functions. These execute in the context of the DPR task. Typically, these callback routines are implemented as simple message posting routines that post messages to an application task. However, the user can choose to implement the indication callback to perform processing within the DPR task context and return without sending any messages. In this case, ensure that the indication routine does not call any API function that changes the driver's state, such as duplexDelete.

The indication routine should be non-blocking because the DPR task executes while interrupts are disabled. The DPR task is also responsible for re-enabling device interrupts once the deferred processing is complete.

Many API functions change the device's state. For information about device states, see page 19.

Figure 7 illustrates the external interfaces defined for the S/UNI-DUPLEX driver.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Figure 7: Application Interface



4.1 Driver Initialization and Shutdown

This section describes the API functions used to initialize and shutdown the driver's modules.

4.1.1 duplexModuleInit: Initializing Driver Modules

This function performs module level initialization of the device driver. This involves allocating memory for the GDD and initializing the data structure.

Valid States	Not applicable		
Side Effects	None		
Prototype	INT4 duplexModuleInit(VOID)		

DRIVER MANUAL PMC-990799 PMC-Sierra, Inc.

PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Inputs	None
Outputs	None
Return Codes	DPX_SUCCESS
	DPX_ERR_MEM_ALLOC (memory allocation failure)
	DPX_ERR_MODULE_ALREADY_INIT

4.1.2 duplexModuleShutdown: Shutting Down Driver Modules

This function performs module level shutdown of the driver. This involves deleting all devices controlled by the driver and deallocating the GDD.

Valid States	All states
Side Effects	Resets all the devices, and removes interrupt handle and DPR task
Prototype	VOID duplexModuleShutdown(VOID)
Inputs	None
Outputs	None
Return Codes	None

4.2 Device Addition, Reset, and Deletion

When you add a new S/UNI-DUPLEX device, the driver's device-addition functions allocate memory to store context information for new devices. The driver also applies a software reset to the device. The device deletion function deallocates device context memory during device shutdown.

4.2.1 duplexAdd: Adding Devices

This function detects the new device in the hardware, gets the base address of the device, and allocates memory for the DDB. Then it stores the device's role (within your system's context) and returns the pointer to the DDB as a handle back to your system. You should use the device handle to identify the device on which the driver will perform the operation.



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

This function also reads the configuration pins status register to determine the configuration of the SCI-PHY/Any-PHY and clocked serial-data interfaces. The driver uses this information to set the device mode in the DDB.

Valid States	DPX_EMPTY		
Side Effects	This function puts the device in the DPX_PRESENT state. The function applies a software reset to the device.		
Prototype	INT4 duplexAdd(DPX_USR_CTXT usrCtxt, DUPLEX *pDuplex)		
Inputs	usrCtxt: Pointer to context information (maintained by your system) for the device being added		
Outputs	 pDuplex: Pointer to the S/UNI-DUPLEX device handle. The variable type, DUPLEX, is actually the following type, which you define: #define DUPLEX (void *) 		
	This prevents the application from accessing the DDB directly.		
Return Codes	DPX_SUCCESS		
	DPX_ERR_INVALID_STATE (invalid device state)		
	DPX_ERR_DEV_NOT_DETECTED (device was not detected)		
	DPX_ERR_MEM_ALLOC (memory allocation failure)		
	DPX_ERR_DEV_ID_TYPE (invalid device ID and/or type)		

4.2.2 duplexReset: Resetting Devices

This function applies a software reset to the S/UNI-DUPLEX device. It also resets all of the device's context information in the DDB (except for the initialization vector, which it leaves unmodified). Typically, the driver calls this function during device shutdown, or before re-initializing the device with an initialization vector.

Valid States All states except DPX_EMPTY

Side Effects This function puts the device in the DPX_PRESENT state. Therefore, the driver must initialize the device after a reset.

DRIVER MANUAL

PIV	IC	PMC-
-----	----	------

Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

РМС-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Prototype	INT4 duplexReset(DUPLEX duplex)	
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX ERR INVALID DEVICE (invalid device	e handle)

4.2.3 duplexRemoteReset: Resetting Other Devices

This function resets other devices by driving the RSTOB output pin of the S/UNI-DUPLEX device low and then back to a high impedance state. It does this by setting and resetting the RESET0 pin in the master configuration register.

Valid States	All states except DPX_EMPTY
Side Effects	None
Prototype	INT4 duplexRemoteReset(DUPLEX duplex)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
Outputs	None
Return Codes	DPX_SUCCESS
	DPX ERR INVALID DEVICE (invalid device handle)

4.2.4 duplexDelete: Deleting Devices

This function removes the specified device from the list of devices controlled by the S/UNI-DUPLEX driver. Deleting a device involves deallocating the DDB for that device.

 Valid States
 DPX_PRESENT

 Side Effects
 This function changes the device state to DPX_EMPTY



DRIVER MANUAL		
PMC-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Prototype	INT4 duplexDelete(DUPLEX duplex)	
Inputs	duplex: Device handle used by the driver to a information for the device (DDB)	access context
Outputs	None	
Return Codes	Return Codes DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device h	nandle)
	DPX ERR INVALID STATE (invalid device st	ate)

4.3 Reading from and Writing to Devices

This section describes the API functions used to read from and write to S/UNI-DUPLEX devices. Their tasks include reading from and writing to the registers of a device.

4.3.1 duplexRead: Reading from Device Registers

This function can read from a register of a specific S/UNI-DUPLEX device by providing the register identifier. This function derives the actual address location based on the device handle and register identifier inputs. It then reads the contents of this address location using the system specific macro, sysDuplexRawRead.

Prototype	INT4 duplexRead(DUPLEX duplex, UINT2 u2RegId, UINT1 *pu1Val)
Inputs	duplex: Pointer to device context information
	u2RegId: Register identifier
Outputs	pulVal: Register value
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_EXCEED_REG_RANGE (u2RegId exceeds the register range)



C. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4.3.2 duplexWrite: Writing to Device Registers

This function can write to a register of a specific S/UNI-DUPLEX device by providing the register identifier. This function derives the actual address location based on the device handle and register identifier inputs. It then writes the contents of this address location using the system specific macro, sysDuplexRawWrite.

Prototype	INT4 duplexWrite(DUPLEX duplex, UINT2 u2RegId, UINT1 u1Val)	
Inputs	duplex: Pointer to device context information	
	u2RegId: Register identifier	
	ulVal: Value to be written	
Outputs	None	
Outputs Return Codes	None DPX_SUCCESS	
Outputs Return Codes	None DPX_SUCCESS DPX_ERR_INVALID_DEVICE (invalid device handle)	

4.4 Device Initialization

This section describes the API functions used to initialize S/UNI-DUPLEX devices. Their tasks include initializing the device based on the initialization vector passed by the application. They also install and remove the indication callback functions that duplexDPR calls.

4.4.1 duplexInit: Initializing Devices

This function initializes the device based on the initialization vector passed by the application. The driver validates this initialization vector and then stores it in the device's DDB. The driver then configures the device registers accordingly.

 Valid States
 DPX_PRESENT

 Side Effects
 This function puts the device in the DPX_INIT state

DRIVER MANUAL PMC-990799



99	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Prototype	INT4 duplexInit(DUPLEX duplex, sInitVector)	sDPX_INIT_VECT,
Inputs	duplex: Pointer to DDB that contains dev maintained by the driver	ice context information
	sInitVector: Initialization vector that the program the device registers	e driver uses to
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid dev	ice handle)
	DPX_ERR_INVALID_STATE (invalid devic	e state)
	DPX_ERR_INVALID_INIT_VECTOR (inva	lid initialization vector)
	DPX_ERR_INDIRECT_CHANNEL_BUSY (Cis busy and causes timeout when its regist	Clocked serial channel ters are accessed)

4.4.2 duplexInstallIndFn: Installing Indication Callback Functions

This function installs the indication callback functions (which you define) that duplexDPR calls. The function pointer is stored in the device context structure (the DDB).

Valid States	DPX_INIT
Side Effects	None
Prototype	INT4 duplexInstallIndFn(DUPLEX duplex, eDPX_CB_TYPE eCbType, DPX_IND_CB_FN pCbFn)

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
	$\tt eCbType$: Identifies the callback being installed, which can be one of:	
	• DPX_CB_NOTIFY	
	• DPX_CB_RX_BOC	
	• DPX_CB_RX_CELL	
	pCbFn: Callback function that the driver is installing	
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	
	DPX_ERR_INVALID_CB_TYPE (invalid callback function type)	

4.4.3 duplexRemoveIndFn: Removing Indication Callback Functions

This function removes the indication callback functions (which you define) that duplexDPR calls.

Valid States	DPX_INIT	
Side Effects	The driver will no longer report events to the application.	
Prototype	INT4 duplexRemoveIndFn(DUPLEX duplex, eDPX_CB_TYPE eCbType)	
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
	$\tt eCbType$: Identifies the callback being installed, which can be one of:	
	• DPX_CB_NOTIFY	
	• DPX_CB_RX_BOC	
	• DPX_CB_RX_CELL	
Outputs	None	

PMC PMC-Sierra, Inc.

C. PM7350 S/UNI-DUPLEX DRIVER

DRIVER MANUAL PMC-990799

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

 Return Codes
 DPX_SUCCESS

 DPX_ERR_INVALID_DEVICE (invalid device handle)

 DPX_ERR_INVALID_CB_TYPE (invalid callback function type)

4.5 Device Activation and Deactivation

This section describes the API functions used to activate and deactivate S/UNI-DUPLEX devices. These functions set the device interrupts and other global enables.

4.5.1 duplexActivate: Activating Devices

This function activates the S/UNI-DUPLEX device by preparing it for normal operation. This involves enabling device interrupts and other global enables (for example, the HSS link transmitter).

Valid States	DPX_INIT
Side Effects	Puts the device in DPX_ACTIVE state.
Prototype	INT4 duplexActivate(DUPLEX duplex)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
Outputs	None
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)

4.5.2 duplexDeactivate: Deactivating Devices

This function de-activates the S/UNI-DUPLEX device and removes it from normal operation. This involves disabling device interrupts and other global disables (for example, the HSS link transmitter).

DRIVER MANUAL



PM7350 S/UNI-DUPLEX DRIVER

PMC-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Valid States	DPX_ACTIVE	
Side Effects	Puts the device in DPX_INIT state.	
Prototype	INT4 duplexDeactivate(DUPLEX dupl	.ex)
Inputs	duplex: Pointer to DDB that contains device maintained by the driver	context information
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device	handle)
	DPX_ERR_INVALID_STATE (invalid device s	state)

4.6 Device Diagnostics

This section describes the API functions used to diagnose the S/UNI-DUPLEX device. Their tasks include:

- Verifying the correctness of the microprocessor's access to the device registers
- Enabling or disabling a diagnostic or line loopback on the HSS interfaces
- Monitoring the activity of the device's clocks

4.6.1 duplexRegisterTest: Verifying Device Register Access

This function verifies the correctness of the microprocessor's access to the device registers by writing values to the writable registers and reading them back.

Valid States	DPX_PRESENT
Side Effects	Puts the device in the DPX_PRESENT state after the test. Therefore, the device should be re-initialized after calling this function.
Prototype	INT4 duplexRegisterTest(DUPLEX duplex)
Inputs	${\tt duplex}$: Pointer to DDB that contains device context information maintained by the driver

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Outputs	None
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)
	DPX_FAILURE (test failed)

4.6.2 duplexLoopback: Enabling/Disabling Diagnostic or Line Loopback

This function enables or disables a diagnostic or line loopback on the HSS interfaces.

Valid States	All states except DPX_EMPTY
Side Effects	None
Prototype	INT4 duplexLoopback(DUPLEX duplex, UINT1 ulHssLnkId, UINT1 ulLpbkType, UINT1 ulEnable)
Inputs	${\tt duplex}:$ Pointer to DDB that contains device context information maintained by the driver
	ulHssLnkId: HSS link identifier. Valid identifiers are DPX_RXD1 and DPX_RXD2.
	ullpbkType: Type of loopback. It can be DPX_DIAG_LPBK or DPX_LINE_LPBK.
	ulEnable: Loopback operation requested. It can be DPX_LPBK_SET or DPX_LPBK_RESET.
Outputs	None

PMC PMC-Sierra, Inc.

PM7350 S/UNI-DUPLEX DRIVER

DRIVER MANUAL PMC-990799

S/UNI-DUPLEX DRIVER MANUAL

Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	
	DPX_ERR_INVALID_STATE (invalid device state)	
	DPX_ERR_INVALID_LPBK_TYPE (invalid loopback type)	
	DPX_ERR_INVALID_HSS_ID (invalid HSS-link identifier)	
	DPX_ERR_INVALID_FLAG (invalid loopback flag)	

4.6.3 duplexGetClockStatus: Monitoring Device Clocks

ISSUE 1

This function monitors the activity of the S/UNI-DUPLEX device clocks. It reads the contents of the clock monitor register and provides the status of each clock in a bit vector format. The application should call this function periodically to check if the clock signals are making low to high transitions.

Valid States	All states except DPX_EMPTY	
Side Effects	None	
Prototype	INT4 duplexGetClockStatus(DUPLEX duplex, UINT1 *pu1ClkStat)	
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
Outputs	pulClkStat: Contains the following bit vector that indicates the active/inactive status of the S/UNI-DUPLEX device clocks. A one in the bit position indicates that the clock is active. A zero indicates that the clock is inactive.	
	 Bit 0: Input FIFO clock (IFCLK) (SCI-PHY/Any-PHY Interface) 	
	 Bit 1: Output FIFO clock (OFCLK) (SCI-PHY/Any-PHY Interface) 	
	Bit 2: Reference clock input (REFCLK)	
Return Codes	S DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4.7 HSS Link Configuration

This section describes the API functions used to configure HSS links. Their tasks include:

- Retrieving the contents of the specified HSS-link's configuration registers
- Configuring or modifying the contents of the specified HSS-link's configuration registers
- Getting a snapshot of the state of the eight HSS links for the specified device
- Retrieving the logical-channel address information for all HSS links of the specified device

4.7.1 duplexHssActiveLnkGetCfg: Getting HSS-Link Selection Method Information

This function obtains information about the active-link selection method configured in the Master Configuration register. This information states whether the active link is set automatically by the S/UNI-DUPLEX, or if it was set manually by the application. If the active link was set manually, then this information states what the manual setting is.

Valid States	DPX_INIT, DPX_ACTIVE	
Side Effects	None	
Prototype	INT4 duplexHssActiveLnkGetCfg(DUPLEX duplex, eHSS_LNK_SEL *peLnkSel)	
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
Outputs	peLnkSel: Specifies the HSS link selection. It can be one of:	
	• DPX_RX_HSS_LNK_SELECT_AUTO	
	• DPX_RX_HSS_LNK_SELECT_RXD1	
	• DPX_RX_HSS_LNK_SELECT_RXD2	
Return Codes	S DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	
	DPX_ERR_INVALID_STATE (invalid device state)	



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4.7.2 duplexHssActiveLnkSetCfg: Setting Active HSS Links

This function sets the active HSS link of the S/UNI-DUPLEX device. The active link can be set automatically by the device or set manually by the application.

Valid States	DPX_INIT, DPX_ACTIVE	
Side Effects	None	
Prototype	INT4 duplexHssActiveLnkSetCfg(DUPLEX duplex, eHSS_LNK_SEL eLnkSel)	
Inputs	${\tt duplex}$: Pointer to DDB that contains device context information maintained by the driver	
	$\tt eLnkSel:$ Specifies the HSS link selection, which can be one of:	
	• DPX_RX_HSS_LNK_SELECT_AUTO	
	• DPX_RX_HSS_LNK_SELECT_RXD1	
	• DPX_RX_HSS_LNK_SELECT_RXD2	
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	
	DPX_ERR_INVALID_STATE (invalid device state)	

4.7.3 duplexHssGetConfig: Getting HSS-Link Configuration Information

This function retrieves the contents of the specified HSS link's configuration registers. With one call, this function can retrieve the value of individual configuration registers as well as the entire configuration register set.

Valid States	DPX_INIT, DPX_ACTIVE
Side Effects	None
Prototype	INT4 duplexHssGetConfig(DUPLEX duplex, eDPX_HSS_REG eHssRegId, sDPX_HSS_REGS *psHssRegs)

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
	eHssRegId: Specifies the register holding the value the driver will retrieve. It can be one of:	
	 DPX_RX_HSS_CFG_RXD0 	
	• DPX_RX_HSS_CFG_RXD1	
	 DPX_RX_HSS_CELL_FILTER_CFG_RXD0 	
	 DPX_RX_HSS_CELL_FILTER_CFG_RXD1 	
	• DPX_TX_HSS_CFG	
	• DPX_ALL_HSS_REGS	
	Note: The logical-channel base address and address range are retrieved together. In addition, the driver can retrieve all configuration registers at once using DPX_ALL_HSS_REGS.	
Outputs	psHssRegs: Contents of the specified HSS-link control register(s) output by this function. These contents are valid only if the function returns DPX_SUCCESS. Further, only those fields of this structure are valid that have been requested using the input parameter, eHssRegId.	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	
	DPX_ERR_INVALID_STATE (invalid device state)	

DPX_ERR_INVALID_REG_ID (invalid register ID)

4.7.4 duplexHssSetConfig: Modifying HSS-Link Configuration Information

This function sets up or modifies the contents of the specified HSS-link's configuration registers. With one call, this function can set the value of individual configuration registers as well as the entire configuration register set.

Valid States DPX_INIT, DPX_ACTIVE

Side Effects None

DRIVER MANUAL

PMC PMC-Sierra, Inc.

Inc. PM7350 S/UNI-DUPLEX DRIVER

PMC-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Prototype	INT4 duplexHssSetConfig(DUPLEX d eDPX_HSS_REG eHssRegId, sDPX_HSS *psHssRegs)	uplex, _REGS
Inputs	duplex: Pointer to DDB that contains devic maintained by the driver	e context information
	eHssRegId: Specifies the register with the write. It can be one of:	value the driver will
	 DPX_RX_HSS_CFG_RXD0 	
	 DPX_RX_HSS_CFG_RXD1 	
	• DPX_RX_HSS_CELL_FILTER_CFG_RXI	00
	• DPX_RX_HSS_CELL_FILTER_CFG_RXI	01
	• DPX_TX_HSS_CFG	
	• DPX_ALL_HSS_REGS	
	Note: The logical channel base address and to be set together. In addition, the driver can registers at once using DPX_ALL_HSS_REG	l address range have a set all configuration s.
	psHssRegs: Contents of the specified HSS register(s) to be set. The only fields in this s set are those that the driver has requested u	-link control tructure that will be using eHssRegId.
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid devic	e handle)
	DPX_ERR_INVALID_STATE (invalid device	state)
	DPX ERR INVALID REG ID (invalid regist	er ID)

4.8 HSS-Link Cell Insertion and Extraction

This section describes the API functions used to insert cells into, and extract cells from, HSS-links. Their tasks include:

- Transmitting a cell on a specified HSS-link 's control channel
- Extracting a cell received on a specified HSS-link 's control channel



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

- Returning the contents of the microprocessor extract FIFO ready register
- Enabling the interrupt indication for a cell's reception
- Installing a callback function that determines the type of cell being extracted

4.8.1 duplexInsertCell: Inserting Cells into HSS Links

This function transmits a cell on the control channel of both the active and standby HSS links. This function can send messages, which you define, over the HSS links. If the message is longer than the length of a cell's payload, then the application should segment the message into 48 byte cells. Call this function repeatedly until all the cells that constitute the message have been transmitted.

Optionally, a 32-bit CRC can protect messages. The CRC accumulates each time a cell belonging to the message is sent. For the last cell of the message (indicated by the application), the CRC is inserted into the last four bytes of the cell's payload.

Message interleaving (over different circuits in the same control channel) is allowed. For CRC-32 protected messages, message interleaving requires the application to save the intermediate CRC-32 value output by this function, if a cell has to be sent out on another control channel or another circuit on the same control channel.

Valid States	DPX_ACTIVE

- Side Effects You should give cell reception higher priority than cell transmission to prevent extract FIFO overflow. In other words, all cells of a received message should be extracted before switching context.
- Prototype INT4 duplexInsertCell(DUPLEX duplex, sDPX_CELL_HDR *psCellHdr, UINT1 *pulCellPyld, sDPX_CELL_CTRL *psCtrl)

DRIVER MANUAL PMC-990799 PMC-Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

S/UNI-DUPLEX DRIVER MANUAL

Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
	psCellHdr: Pointer to the cell header structure that contains the two prepend bytes that you define (optional), H1-H4 bytes, and the H5 (optional) and UDF (optional) bytes. The driver uses the optional bytes based on the TX HSS configuration register contents.
	pulCellPyld: Pointer to first byte of cell payload (48 contiguous bytes)
	psCtrl->u1CrcFlg: Control flag containing the following bit vectors:
	 Bit 0: Flag for CRC protection flag Bit 1: Flag for first cell of a CRC protected message Bit 2: Flag for last cell of a CRC protected message
	psCtrl->u4Crc32Prev: Used to restore previously saved CRC-32 value output by this function. Only applicable if bit 0 of psCtrl->u1CrcFlg is set.
Outputs	psCtrl->u4Crc32: Used to output CRC-32 value after writing a cell. The driver then passes this value back as an input parameter (psCtrl->u4Crc32Prev) for the next cell to be transmitted on the same control channel connection.
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)
	DPX_ERR_CELL_TX_BUSY (cell transmission port busy)

4.8.2 duplexExtractCell: Extracting Cells from HSS Links

ISSUE 1

This function extracts a cell received on a specified HSS-link 's control channel. This function also receives messages, which you define, that can span multiple cells. The application must call this function once for each cell that constitutes the message.



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

If the incoming message contains a CRC-32 field at the end, then the driver can perform a CRC check over the body of the message. The function also provides the header information of the cell to the calling function.

Valid States	DPX_ACTIVE
Side Effects	You should give cell reception a higher priority than cell transmission to prevent extract FIFO overflow. In other words, all cells of a received message should be extracted before switching context.
Prototype	INT4 duplexExtractCell(DUPLEX duplex, UINT1 ulHssLnkId, sDPX_CELL_HDR *psCellHdr, UINT1 *pulCellPyld, sDPX_CELL_CTRL *psCtrl)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
	ulHssLnkId: HSS link identifier. Valid identifiers are DPX_RXD1 and DPX_RXD2.
Outputs	psCellHdr: Pointer to the cell header-data received
	pulCellPyld: Pointer to first byte of cell payload 48 contiguous bytes)
	psCtrl->u4Crc32: Used to output CRC-32 value after reading a cell. The driver then passes this value back as an input parameter (psCtrl->u4Crc32Prev) for the next cell to be extracted on the same control channel connection.
	psCtrl->u1CrcFlg: This is a control flag. Contains the following bit vector:
	Bit 0: CRC protection flag
	Bit 1: Flag for first cell of a CRC protected message

• Bit 2: Flag for last cell of a CRC protected message

DRIVER MANUAL

PMC PMC-Sierra, Inc.

. PM7350 S/UNI-DUPLEX DRIVER

PMC-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid dev	vice handle)
	DPX_ERR_INVALID_STATE (invalid devi	ce state)
	DPX_ERR_INVALID_LNK_ID (invalid link	< ID)
	DPX_ERR_CB_FN_NOT_INSTALLED (cal installed yet)	lback function is not
	DPX_ERR_CELL_RX_CRC (cell reception	CRC error)

4.8.3 duplexCheckExtractFifos: Getting Contents of the Extract-FIFO-Ready Register

This function returns the contents of the microprocessor extract-FIFO-ready register. This function can check if there are any cells to extract from the extract FIFOs.

Valid States	DPX_ACTIVE
Side Effects	None
Prototype	UINT4 duplexCheckExtractFifos(DUPLEX duplex, UINT1 *pulCellReady)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
Outputs	pulCellReady: Contains the following bit vector, which represents the state of each extract FIFO:
	• Bit 0:
	• If value is 1, then RXD1 has at least one cell ready for extraction
	 If value is 0, then no cells present
	• Bit 1:
	 If value is 1, then RXD2 has at least one cell ready for extraction
	 If value is 0, then no cells present

PRELIMINARY DRIVER MANUAL



PM7350 S/UNI-DUPLEX DRIVER

РМС-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device	handle)
	DPX_ERR_INVALID_STATE (invalid device s	state)

4.8.4 duplexEnableRxCellInd: Enabling the Received Cell Indicator

This function enables the interrupt indication in the device for the reception of a cell. The application calls this function after it has responded to a previous indication by extracting all received cells (using multiple duplexExtractCell calls). The application task can now re-enable this indication and wait for the arrival of more cells.

Valid States	DPX_ACTIVE
Side Effects	None
Prototype	<pre>INT4 duplexEnableRxCellInd(DUPLEX duplex)</pre>
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
Outputs	None
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)

4.8.5 duplexInstallCellTypeFn: Installing Callback Functions

This function can install a callback function (which you define) that the driver uses to determine the type of cell it is extracting. The detector function takes a cell header as the input argument and returns a cell type byte and the previous CRC-32 value for the same message of the same logical channel.

Valid States DPX_INIT, DPX_ACTIVE

Side Effects None

DRIVER MANUAL

PMC PMC-Sierra, Inc.

Inc. PM7350 S/UNI-DUPLEX DRIVER

РМС-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Prototype	duplexInstallCellTypeFn(DUPLEX o DPX_EOM_FN pCellTypeFn)	duplex,
Inputs	duplex: Pointer to DDB that contains devi information maintained by the driver	ce context
	pCellTypeFn: Pointer to the EOM detector prototype of this function is:	or function. The
	 UINT1 pCellTypeFn(UINT1 *pulk *pu4Crc32Prev) 	Idr, UINT4
	In the detector function, pulHdr is the po of the cell header's eight bytes. pu4Crc32 accumulated CRC for the previous cells re- message.	inter to the first byte Prev is the ceived for the same
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid devi	ce handle)
	DPX_ERR_INVALID_STATE (invalid device	e state)

4.9 BOC Transmission and Reception

This section describes the API functions used to transmit and receive bit-oriented code (BOC). Their tasks include transmitting the specified BOC on the specified HSS link, and reading the BOC received on a HSS link

4.9.1 duplexTxBOC: Transmitting BOC

This function transmits the specified BOC on the specified HSS link. In the case of transmitting a loopback activate-BOC code, the RDIDIS register bit should be set to logic 1 before the transmission. This prevents a pre-emptive remote-defect-indication (RDI) code from being sent.

Valid States	DPX_ACTIVE
Side Effects	None
Prototype	INT4 duplexTxBOC(DUPLEX duplex, UINT1 ulHssLnkId, UINT1 ulCode)

DRIVER MANUAL PMC-990799



DPX_ERR_INVALID_STATE (invalid device state)

DPX_ERR_INVALID_LNK_ID (invalid link ID)

DPX_ERR_INVALID_BOC (invalid BOC)

4.9.2 duplexRxBOC: Reading from Received BOC

This function can read BOC received on a HSS link.

Valid States	DPX_ACTIVE
Side Effects	This function reads from the receive-BOC status register. The

application should call this function inside the indDuplexRxBOC indication-callback function. This function clears the status bits (IDLEI and BOCI) in the BOC status register.

PrototypeINT4 duplexRxBOC(DUPLEX duplex, UINT1
ulHssLnkId, UINT1 *pulCode)

DRIVER MANUAL

PMC-Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

РМС-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Inputs	duplex: Pointer to DDB that contains device information maintained by the driver ulHssLnkId: HSS link identifier. Valid iden DPX_RXD1 and DPX_RXD2.	e context tifiers are
Outputs	 pulCode: Pointer to BOC to be received. Value 000000b (RDI) 000001b (loopback activate) 000010b (loopback deactivate) 000011b (remote reset activate) 000100b (remote reset deactivate) 010001b to 111110b (defined by you) 111111b (idle code) 	alid BOCs are:
Return Codes	DPX_SUCCESS DPX_ERR_INVALID_DEVICE (invalid device DPX_ERR_INVALID_STATE (invalid device DPX_ERR_INVALID_LNK_ID (invalid link line) DPX_ERR_INVALID_BOC (invalid BOC)	e handle) state) D)

4.9.3 duplexSetAutoRDITx: Transmitting Remote-Defect Indication Code Words

Enables/disables the automatic transmission of an RDI code word on the specified HSS link.

Valid States	DPX_INIT, DPX_ACTIVE
Side Effects	None
Prototype	<pre>INT4 duplexSetAutoRDITx(DUPLEX duplex, UINT1 ulHssLnkId, UINT1 ulDisableFlg)</pre>

DRIVER MANUAL

PMC PMC-Sierra, Inc.

C. PM7350 S/UNI-DUPLEX DRIVER

РМС-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Inputs	duplex: Pointer to DDB that contains information maintained by the driver	s device context
	ulHssLnkId: HSS link identifier. Vali DPX_RXD1 and DPX_RXD2.	d identifiers are
	u1DisableF1g: 1 enables auto trans disables auto transmission of RDI.	smission of RDI. 0
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid	d device handle)
	DPX_ERR_INVALID_STATE (invalid)	device state)

DPX_ERR_INVALID_LNK_ID (invalid link ID)

4.9.4 duplexSciAnyPhyGetConfig: Getting SCI-PHY/Any-PHY Configuration Information

This function retrieves the contents of the S/UNI-DUPLEX SCI-PHY/Any-PHY configuration registers. It can retrieve the value of individual configuration registers. Alternatively, it can retrieve the entire configuration register set with one call.

Valid States	DPX_INIT, DPX_ACTIVE
Side Effects	None
Prototype	<pre>INT4 duplexSciAnyPhyGetConfig(DUPLEX duplex, eDPX_SCI_ANY_PHY_REG eSciAnyPhyRegId, sDPX_SCI_ANY_PHY_REGS *psSciAnyPhyRegs)</pre>

DRIVER MANUAL PMC-990799

PMC-Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
	eSciAnyPhyRegId: Specifies the register containing the value to be retrieved. It can be one of:
	• DPX_SCI_ANY_PHY_EXT_ADDR_MATCH
	 DPX_SCI_ANY_PHY_EXT_ADDR_MASK
	• DPX_SCI_ANY_PHY_OUT_ADDR_MATCH
	 DPX_SCI_ANY_PHY_INP_CFG_1
	• DPX_SCI_ANY_PHY_INP_CFG_2
	• DPX_SCI_ANY_PHY_ICA_ENBL_LSB
	• DPX_SCI_ANY_PHY_ICA_ENBL_2
	• DPX_SCI_ANY_PHY_ICA_ENBL_3
	• DPX_SCI_ANY_PHY_ICA_ENBL_MSB
	• DPX_SCI_ANY_PHY_OUT_CFG
	• DPX_SCI_ANY_PHY_OUT_POLL_RNG
	• DPX_ALL_PHY_REGS
	Note: All configuration registers can be retrieved at once using DPX_ALL_PHY_REGS
Outputs	psSciAnyPhyRegs: Contents of the specified SCI-PHY/Any-PHY registers output by this function
	These contents are valid only if the function returns DPX_SUCCESS. Also, the only fields in this structure that are valid are those that have been requested using the input parameter, eSciAnyPhyRegId.
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)
	<pre>DPX_ERR_INVALID_REG_ID (invalid register ID)</pre>



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4.9.5 duplexSciAnyPhySetConfig: Configuring HSS-Links

This function configures and modifies the contents of the specified HSS-link's configuration registers. It can set the value of individual configuration registers. Alternatively, it can set the entire configuration register set with one call.

Valid States	DPX_INIT, DPX_ACTIVE
Side Effects	None
Prototype	INT4 duplexSciAnyPhySetConfig(DUPLEX duplex, eDPX_SCI_ANY_PHY_REG eSciAnyPhyRegId, sDPX_SCI_ANY_PHY_REGS *psSciAnyPhyRegs)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver.
	eSciAnyPhyRegId: Specifies the register containing the value to be set; can be one of:
	• DPX_SCI_ANY_PHY_EXT_ADDR_MATCH
	• DPX_SCI_ANY_PHY_EXT_ADDR_MASK
	• DPX_SCI_ANY_PHY_OUT_ADDR_MATCH
	• DPX_SCI_ANY_PHY_INP_CFG_1
	• DPX_SCI_ANY_PHY_INP_CFG_2
	• DPX_SCI_ANY_PHY_ICA_ENBL_LSB
	• DPX_SCI_ANY_PHY_ICA_ENBL_2
	• DPX_SCI_ANY_PHY_ICA_ENBL_3
	• DPX_SCI_ANY_PHY_ICA_ENBL_MSB
	• DPX_SCI_ANY_PHY_OUT_CFG
	• DPX_SCI_ANY_PHY_OUT_POLL_RNG
	• DPX_ALL_PHY_REGS
	Note: All configuration registers can be set at once using DPX_ALL_PHY_REGS.
	psSciAnyPhyRegs: Contents of the specified SCI-PHY/Any-PHY registers that this function will set. These

SCI-PHY/Any-PHY registers that this function will set. These contents are valid only if the function returns DPX_SUCCESS. Also, only those fields of this structure are valid that have been requested using the input parameter, eSciAnyPhyRegId.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid devic	e handle)
	DPX_ERR_INVALID_STATE (invalid device	state)
	DPX_ERR_INVALID_REG_ID (invalid regist	er ID)

4.10 Clocked Serial-Data Interface Functions

The clocked serial-data interface functions perform the following tasks:

- Reads from transmit and receive serial-channel context bytes
- Writes to transmit and receive serial-channel context bytes
- Enables and disables automatic reset of the HCS error-count register

4.10.1 duplexRxSerChnIReadReg: Reading from Receive Serial-Channel Context Bytes

This function indirectly reads a receive serial-channel context byte.

Valid States	DPX_INIT, DPX_PRESENT
Side Effects	None
Prototype	INT4 duplexRxSerChnlReadReg(DUPLEX duplex, UINT1 ulSerChnlId, eDPX_CLK_SER_REG eClkSerRegId, UINT1 *pulRegVal)

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver ulserChnlld: Serial channel identifier (0 through 15)
	eClkSerRegId: Specifies the register containing the value that the function will retrieve. It can be one of the following:
	• DPX_CLK_SER_RX_CHNL_CFG
	• DPX_CLK_SER_RX_INT_ENBLS
	• DPX_CLK_SER_RX_INT_STATUS
	• DPX_CLK_SER_RX_HCS_ERR_CNT
	• DPX_CLK_SER_LCD_CNT_THRESH
Outputs	pulRegVal: Contents of the specified clocked-bit serial-interface registers output by this function. These contents are valid only if the function returns DPX_SUCCESS.
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)
	DPX_ERR_INVALID_CHNL_ID (invalid serial channel ID)
	DPX_ERR_INVALID_REG_ID (invalid register ID)

4.10.2 duplexRxSerChnlWriteReg: Writing to Receive Serial-Channel Context Bytes

This function indirectly writes to a receive serial-channel context byte.

Valid States	DPX_INIT, DPX_PRESENT
Side Effects	None
Prototype	<pre>INT4 duplexRxSerChnlWriteReg(DUPLEX duplex, UINT1 ulSerChnlId, eDPX_CLK_SER_REG eClkSerRegId, UINT1 ulRegVal)</pre>
DRIVER MANUAL PMC-990799 PMC-Sierra, Inc.

ISSUE 1

Inc. PM7350 S/UNI-DUPLEX DRIVER

S/UNI-DUPLEX DRIVER MANUAL

Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver ulSerChnlld: Serial channel identifier (0 through 15)
	eClkSerRegId: Specifies the register containing the value that the function will retrieve. It can be one of the following:
	• DPX_CLK_SER_RX_CHNL_CFG
	• DPX_CLK_SER_RX_INT_ENBLS
	• DPX_CLK_SER_LCD_CNT_THRESH
	ulRegVal: Contents of the specified clocked-bit serial-interface register that this function will set
Return Codes	DPX_SUCCESS
	DPX_ERR_INVALID_DEVICE (invalid device handle)
	DPX_ERR_INVALID_STATE (invalid device state)
	DPX_ERR_INVALID_CHNL_ID (invalid serial channel ID)
	<pre>DPX_ERR_INVALID_REG_ID (invalid register ID)</pre>

4.10.3 duplexRxSerChnIHCSCntResetEn: Enabling Auto Reset of HCS Error Registers

This function enables or disables automatic reset of the HCS error-count register when an indirect read is initiated.

Valid States	DPX_INIT, DPX_ACTIVE
Side Effects	None
Prototype	INT4 duplexRxSerChnlHCSCntResetEn(DUPLEX duplex, UINT1 ulEnable)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
	ulEnable: If value is 0, the flag enables auto reset. If the value is not 0, the flag disables autoreset.

DRIVER MANUAL



PM7350 S/UNI-DUPLEX DRIVER

РМС-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device	e handle)
	DPX_ERR_INVALID_STATE (invalid device	state)

4.10.4 duplexTxSerChnIReadReg: Reading from Transmit Serial-Channel Context Bytes

This function indirectly reads a transmit serial-channel context byte.

Valid States	DPX_INIT, DPX_PRESENT	
Side Effects	None	
Prototype	INT4 duplexTxSerChnlReadReg(DUPLEX duplex, UINT1 ulSerChnlId, eDPX_CLK_SER_REG eClkSerRegId, UINT1 *pulRegVal)	
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
	ulSerChnlId: Serial channel identifier (0 through 15)	
	eClkSerRegId: Specifies the register containing the value that this function will retrieve. Its value can be one of the following:	
	• DPX_CLK_SER_TX_DATA	
	• DPX_CLK_SER_TX_SER_FRM_BIT_THRESH	
Outputs	pulRegVal: Contents of the specified clocked-bit serial-interface register output by this function. These contents are valid only if the function returns DPX_SUCCESS	

DRIVER MANUAL

PMC-Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

PMC-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid dev	vice handle)
	DPX_ERR_INVALID_STATE (invalid devi	ce state)
	DPX_ERR_INVALID_CHNL_ID (invalid set	erial channel ID)
	DPX_ERR_INVALID_REG_ID (invalid reg	jister ID)

4.10.5 duplexTxSerChnlWriteReg: Writing to Transmit Serial-Channel Context Bytes

This function indirectly writes to a transmit serial-channel context byte.

Valid States	DPX_INIT, DPX_PRESENT		
Side Effects	None		
Prototype	INT4 duplexTxSerChnlWriteReg(DUPLEX duplex, UINT1 u1SerChnlId, eDPX_CLK_SER_REG eClkSerRegId, UINT1 u1RegVal)		
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver		
	ulSerChnlld: Serial channel identifier (0 through 15)		
	eClkSerRegId: Specifies the register containing the value that this function will retrieve. It can be one of the following:		
	• DPX_CLK_SER_TX_DATA		
	• DPX_CLK_SER_TX_SER_FRM_BIT_THRESH		
	u1RegVa1: Contents of the specified clocked-bit serial-interface register that this function will set		

DRIVER MANUAL PMC-990799

C-Sierra, Inc. PM7350 S/UNI-DUPLEX DRIVER

1799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid de	evice handle)
	DPX_ERR_INVALID_STATE (invalid dev	vice state)
	DPX_ERR_INVALID_CHNL_ID (invalid	serial channel ID)
	DPX_ERR_INVALID_REG_ID (invalid re	egister ID)

4.11 Statistics Collection

This section describes the API functions used to collect statistics about the device's HSS links. Their tasks include:

- Accumulating the received-cell count and header-check sequence (HCS) cell-error count for a specified HSS link
- Accumulating the transmitted-cell count for a specified HSS link
- Reading all the cell counts (transmit and receive) for all the HSS links of the specified device
- Retrieving and resetting the statistical counts maintained by the driver

4.11.1 duplexGetHssLnkRxCounts: Accumulating Counts for Received Cells

This function accumulates the counts for received cells and errored HCS cells for a specified HSS link (RXD1 or RXD2). It triggers an update of the receive HSS cell-counter registers and the receive-HSS HCS error-count register. It then reads the contents of these registers and returns the values read to the application. To maintain a steady count, without overflow, of received cells and HCS cell errors, the application should call this function at least every 30 seconds.

Valid States	DPX_ACTIVE
Side Effects	You should not use this function at the same time (in periodic polling fashion) as duplexGetAllHssLnkCounts because both functions trigger updates to the receive counters.
Prototype	<pre>INT4 duplexGetHssLnkRxCounts(DUPLEX duplex, UINT1 u1HssLnkId, UINT4 *pu4RxCells, UINT4 *pu4HcsErrs)</pre>

PRELIMINARY		PMC-Sierra, Inc.	PM7350 S/UNI-DUPLEX DRIVER
DRIVER MANUAL PMC-990799	ISSUE 1		S/UNI-DUPLEX DRIVER MANUAL
Inputs	duplex: Pointer to information maintair	:: Pointer to DDB that contains device context tion maintained by the driver	
	ulHssLnkId: HSS DPX_RXD1 and DPX	link identifier. Valic	l identifiers are
Outputs	pu4RxCells: Cour	nt of cells received	
	pu4HcsErrs: Cour	nt of errored HCS-o	cells received
Return Codes	Return Codes DPX_SUCCESS		
	DPX_ERR_INVALII	D_DEVICE (invalid	device handle)
	DPX_ERR_INVALII	D_STATE (invalid d	levice state)
	DPX_ERR_INVALII	D_LNK_ID (invalid	link ID)

4.11.2 duplexGetHssLnkTxCounts: Accumulating Counts for Transmitted Cells

This function accumulates the counts for transmitted cells for a specified HSS link (TXD1 or TXD2). It triggers an update of the transmit HSS cell-counter registers. It then reads the contents of these registers and returns the values read to the application. To maintain a steady count, without overflow, of transmitted cells, the application should call this function at least every 30 seconds.

Valid States	DPX_ACTIVE
Side Effects	You should not use this function at the same time (in periodic polling fashion) as duplexGetAllHssLnkCounts because both functions trigger updates to the transmit counters.
Prototype	INT4 duplexGetHssLnkTxCounts(DUPLEX duplex, UINT4 *pu4TxCells)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
Outputs	pu4TxCells: Count of cells transmitted

DRIVER MANUAL



PM7350 S/UNI-DUPLEX DRIVER

PMC-990799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid devi	ce handle)
	DPX_ERR_INVALID_STATE (invalid devic	e state)

4.11.3 duplexGetAllHssCounts: Accumulating Counts for All Cells

This function reads all the cell counts (transmit and receive) for all the serial links of the specified S/UNI-DUPLEX device. This function triggers an update to the RXD1 and RXD2 receive and transmit counters by writing a dummy value to the load performance meters register. It then reads the counters of all the serial links and returns the contents to the calling function.

To maintain a steady count of cells received, cells transmitted, and HCS errored cells on a per-link basis for all the serial links, and to avoid overflow, the application should call this function at least every 30 seconds.

Valid States	DPX_ACTIVE	
Side Effects	You should not use this function at the same time (in periodic polling fashion) as duplexGetHssLnkRxCounts and duplexGetHssLnkTxCounts because both functions trigger updates to the same counters.	
Prototype	INT4 duplexGetAllHssCounts(DUPLEX duplex, sDPX_HSS_CNTS *psHssCnts)	
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver	
Outputs	psHssCnts: Contains the RXD1 and RXD2 cells received, errored received cells, and transmitted cells	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	
	DPX_ERR_INVALID_STATE (invalid device state)	



S/UNI-DUPLEX DRIVER MANUAL

4.11.4 duplexGetStatisticCounts: Retrieving Driver Statistical Counts

This function retrieves the statistical counts maintained by the driver. It contains the counts for events and interrupts of the S/UNI-DUPLEX device since the last call to reset statistic counts.

Valid States	All states except DPX_EMPTY	
Side Effects	None	
Prototype	INT4 duplexGetStatisticCounts(DUPLEX duplex, sDPX_STAT_COUNTS *psStatCounts)	
Inputs	duplex: Pointer to DDB that contains the count information maintained by the driver	
Outputs	psStatCounts: Contains statistical counts of events and interrupts	
Return Codes	DPX_SUCCESS	
	DPX ERR INVALID DEVICE (invalid device handle)	

4.11.5 duplexResetStatisticCounts: Resetting Driver Statistical Counts

This function resets the statistical counts maintained by the driver.

Valid States	All states except DPX_EMPTY	
Side Effects	None	
Prototype	<pre>INT4 duplexResetStatisticCounts(DUPLEX duplex)</pre>	
Inputs	duplex: Pointer to DDB that contains the count information maintained by the driver	
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INVALID_DEVICE (invalid device handle)	

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

4.12 Indication Callbacks

The DPR uses indication callback functions to notify the application of events in the S/UNI-DUPLEX device and driver. You must implement these functions to work within the inter-task communication and scheduling capabilities of your RTOS. Typically, the callback functions will run in the context of the DPR, not in the context of the application. Therefore, these functions must be non-blocking. They should use RTOS-based inter-task notification to pass callback information safely from the DPR to the application task.

4.12.1 indDuplexNotify: Notifying the Application of Significant Events

This indication function notifies the application about the occurrence of a significant event in the hardware or the driver software. The duplexDPR function calls this function. This function should be non-blocking. Typically, the indication function sends a message to another task with the event identifier and other context information. The task that receives this message can then process this information according to the system requirements.

Prototype	VOID indDuplexNotify(USR_CTXT usrCtxt, sDPX_IND_BUF *pIndBuf)
Inputs	$\tt usrCtxt:$ Context information (maintained by your system) for the device
	pIndBuf: Information regarding the indication. It consists of an event identifier that identifies the reported event. Uniquely supplemental information about the event. The application should use duplexReturnIndBuf to free the indication context structure.
Outputs	None
Return Codes	None

4.12.2 indDuplexRxBOC: Notifying the Application of Received BOC

This indication function notifies the application about the reception of a valid BOC. The duplexDPR function calls this function. This function should be non-blocking.

DRIVER MANUAL PMC-990799

IARY	PMC-Sierra, Inc.	PM7350 S/UNI-DUPLEX DRIVER	
799	ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL	
Inputs	usrCtxt: Context information (maintained by your system) for the device		
	pIndBuf: Information regarding the indic	cation. It consists of:	
	• ulHssLnkId: HSS link that received	the BOC.	
	• u1BOC: BOC received. It can be one	of the following:	
	• 000000b (RDI)		
	000001b (loopback activate)		
	 000010b (loopback deactivate) 000011b (remote reset activate) 		
	 000100b (remote reset deactivate) 	.)	
	 010001b to 111110b (defined by y 	you)	
	 111111b (idle code) 		

The application should use duplexReturnIndBuf to free the indication context structure.

Outputs None

Return Codes None

4.12.3 indDuplexRxCell: Notifying the Application of Ready Extract-Cell-FIFOs

This indication function notifies the application of the reception of cells in the microprocessor extract cell FIFOs. The duplexDPR function calls this function. This function should be non-blocking. Typically, the indication function sends a message to another task with the event identifier and other context information. The task that receives this message can then extract the received cells using duplexCheckExtractFifos and duplexExtractCell.

Prototype	VOID indDuplexRxCell(USR_CTXT usrCtxt, sDPX_IND_BUF *pIndBuf)	
Inputs	usrCtxt: Context information (maintained by your system) for the device pIndBuf: Information regarding the indication. Currently, the driver does not use it, so the driver passes a null pointer for now.	
Outputs	None	
Return Codes	None	

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

5 REAL-TIME-OS INTERFACE FUNCTIONS

The driver's RTOS interface module provides functions and macros that let the driver use RTOS services. The S/UNI-DUPLEX driver requires the following RTOS services:

- Memory: Allocate and deallocate
- Interrupts: Install and remove
- Preemption: Enable and disable

The driver may also require the following additional RTOS services depending on how you customize the code (for example, the ISR, the DPR, and so on). These services are:

- Timers: Create, delete, start and abort
- Tasks: Spawn and delete
- Message queues: Create and destroy queues, send and receive messages

Figure 8 illustrates the external interfaces defined for the S/UNI-DUPLEX driver.

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Figure 8: Real-Time OS Interface



5.1 Memory Allocation and Deallocation

This section describes the RTOS interface functions used to allocate and deallocate memory.

5.1.1 sysDuplexMemAlloc: Allocating Memory

This macro allocates a specified number of bytes.

- Prototype #define sysDuplexMemAlloc(nbytes) malloc(nbytes)
- Inputs nbytes: Number of bytes to be allocated

Outputs None

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Return CodesPointer to first byte of allocated memoryNULL pointer (memory allocation failed)

5.1.2 sysDuplexMemFree: Deallocating Memory

This macro deallocates memory allocated by sysDuplexMemAlloc.

Prototype	<pre>#define sysDuplexMemFree(pulFirst) free(pulFirst)</pre>
Inputs	pulFirst: Pointer to first byte of the memory region being deallocated
Outputs	None
Return Codes	None

5.2 Buffer Management

This section describes the RTOS interface functions used to manage buffers for the DPR. Their tasks include getting a buffer for saving the context information for the indication callbacks, and returning the buffer after the application has received the context information.

5.2.1 duplexGetIndBuf: Getting DPR Buffers

This function gets a buffer that saves the context information for the indication callbacks called by the DPR.

Prototype	<pre>sDPX_IND_BUF *duplexGetIndBuf(VOID)</pre>	
Inputs	None	
Outputs	None	
Return Codes	Pointer to indication context buffer	
	NULL pointer (buffer unavailable)	



S/UNI-DUPLEX DRIVER MANUAL

5.2.2 duplexReturnIndBuf: Returning DPR Buffers

This function returns the indication context buffer after the DPR has received the context information.

Prototype	<pre>VOID duplexReturnIndBuf(sDPX_IND_BUF *pBuf)</pre>
Inputs	pBuf: Pointer to indication context structure
Outputs	None
Return Codes	None

5.3 Timer Operations

This section describes the RTOS interface function used to suspend a task for a specified period.

5.3.1 sysDuplexDelayFn: Delaying Functions

This function suspends execution of the calling function's task for a specified number of milliseconds.

Prototype VOID sysDuplexDelayFn(UINT4 u4Msecs)

Inputs u4Msecs: Delay (in milliseconds)

Outputs None

Return Codes None

5.4 Semaphore Operations

This section describes the RTOS interface macros used to manage semaphores. Their tasks include:

- Creating a new mutual-exclusion semaphore
- Deleting a specified semaphore
- Taking and giving semaphores



S/UNI-DUPLEX DRIVER MANUAL

5.4.1 sysDuplexSemCreate: Creating Semaphores

This macro creates a new mutual-exclusion semaphore.

Prototype	<pre>#define sysDuplexSemCreate() semMCreate()</pre>
Inputs	None
Outputs	None
Return Codes	semaphore ID

5.4.2 sysDuplexSemDelete: Deleting Semaphores

This macro deletes a specified semaphore.

Prototype	<pre>#define sysDuplexSemDelete(semId) semDelete(semId)</pre>
Inputs	semaphore ID
Outputs	None
Return Codes	None

5.4.3 sysDuplexSemTake: Taking Semaphores

This macro takes a semaphore.

- Prototype #define sysDuplexSemTake(semId) semTake(semId)
- Inputs semaphore ID
- Outputs None
- Return Codes None

5.4.4 sysDuplexSemGive: Giving Semaphores

This macro gives a semaphore.

DRIVER MANUAL PMC-990799



ISSUE 1	S/UNI-DUPLEX DRIVER MANUAL

Prototype	<pre>#define sysDuplexSemGive(semId) semGive(semId)</pre>
Inputs	semaphore ID
Outputs	None
Return Codes	None

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

6 HARDWARE INTERFACE FUNCTIONS

The S/UNI-DUPLEX hardware interface provides functions and macros that read from and write to S/UNI-DUPLEX device-registers. The hardware interface also provides a template for an ISR that the driver calls when the device raises a hardware interrupt. You must modify this function based on the interrupt configuration of your system.

Figure 9 illustrates the external interfaces defined for the S/UNI-DUPLEX driver.

Figure 9: Hardware Interface



6.1 Device Register Access

This section describes the hardware interface functions used to read from and write to S/UNI-DUPLEX device registers. Their tasks include reading and writing the contents of a specific address. It also includes getting the base address of the new device so that the driver can access the device register map to control it.



S/UNI-DUPLEX DRIVER MANUAL

6.1.1 sysDuplexRawRead: Reading from Register Address Locations

This low-level system-specific macro reads the contents of a specific register-address location. You should define this to reflect your system's addressing logic.

Prototype#define sysDuplexRawRead(addr, val)Inputsaddr: Address location to be readOutputsval: Value read

6.1.2 sysDuplexRawWrite: Writing to Register Address Locations

This low-level system-specific macro writes the contents of a specific register-address location. You should define this macro to reflect your system's addressing logic.

Prototype	<pre>#define sysDuplexRawWrite(addr, val)</pre>
Inputs	addr: Address location to write
	val: Value to be written

Outputs None

6.1.3 sysDuplexDeviceDetect: Getting Device Base Addresses

This function uses user context information to detect new S/UNI-DUPLEX devices. The duplexAdd API function calls it. This function's implementation is system specific.

Prototype	<pre>INT4 sysDuplexDeviceDetect(DPX_USR_CTXT usrCtxt, VOID **ppSysInfo, UINT4 *pu4BaseAddr)</pre>
Inputs	usrCtxt: Context information (maintained by your system) for the device



ISSUE 1

PM7350 S/UNI-DUPLEX DRIVER

DRIVER MANUAL PMC-990799

S/UNI-DUPLEX DRIVER MANUAL

Outputs	pu4BaseAddr: Base address of device
	ppSysInfo: Pointer to a system information buffer, which contains system specific information
Return Codes	DPX_SUCCESS
	DPX DEVICE NOT DETECTED

6.2 Interrupt Servicing

This section describes the hardware interface functions used to provide hardware interrupt servicing. They install and remove the interrupt handlers and DPRs for the S/UNI-DUPLEX devices. These functions depend on whether you implement the driver in interrupt mode or polling mode. In interrupt mode, their tasks include:

- Installing and removing the system-dependent interrupt-handler function (sysDuplexIntHandler) and the DPR function (sysDuplexDPRTask), creating a communication channel between the two, and adding the device to a list of devices for which interrupts will be serviced
- Removing the specified device from the list of devices for which interrupt processing will be done
- Calling duplexISR for each device for which interrupt processing is enabled
- Retrieving interrupt status information saved for it by the sysDuplexIntHandler function, and calling the duplexDPR function for the appropriate device

In polling mode, these functions' tasks include:

- Spawning and removing the sysDuplexDPRTask function
- · Adding the device to a list of devices that need polling
- Polling the S/UNI-DUPLEX device for interrupt status information and processing the interrupt status

The S/UNI-DUPLEX driver provides a function called duplexISR that checks if there are any valid interrupt conditions present for a specified device. This function can be used by a system-specific interrupt-handler function to service interrupts raised by S/UNI-DUPLEX devices.



S/UNI-DUPLEX DRIVER MANUAL

The low-level interrupt handler function that traps the hardware interrupt and calls duplexISR is system and RTOS dependent. Therefore, it is outside the scope of the driver. As a reference, this manual provides an example implementation of such an interrupt handler (see sysDuplexIntHandler) as well as installation and removal functions (see sysDuplexIntInstallHandler and sysDuplexIntRemoveHandler). You can customize these example implementations as per your specific requirements.

6.2.1 duplexISR: Registering Interrupt Statuses

This function reads the top-level interrupt-status registers of the interrupting device. If there are any bits set in these registers, this function returns a value greater than zero. If there are no bits set, this function returns a zero. This function is invoked by the system-specific interrupt-handler function, sysDuplexIntHandler.

Note: In polling mode, the driver does not use this function.

Valid States	DPX_ACTIVE	
Side Effects	If the function returns with a non-zero value (meaning interrupt conditions have been detected), then all device interrupts are disabled.	
Prototype	UINT4 duplexISR(DUPLEX duplex)	
Inputs	duplex : Pointer to DDB that contains device context information maintained by the driver	
Outputs	None	
Return Codes	= 0 (no valid interrupt conditions detected)	
	> 0 (at least one valid interrupt condition detected)	

6.2.2 duplexDPR: Processing Interrupts

This function performs the following tasks:

- Reads the device interrupt-status registers
- Clears the interrupt conditions
- Invokes user-defined callback functions that perform system-specific processing based on the interrupt conditions detected



S/UNI-DUPLEX DRIVER MANUAL

The system-specific DPR-task function, sysDuplexDPRTask, invokes this function.

Valid States	DPX_ACTIVE
Side Effects	Enables device interrupts after processing all the existing interrupt conditions
Prototype	VOID duplexDPR(DUPLEX duplex)
Inputs	duplex: Pointer to DDB that contains device context information maintained by the driver
Outputs	None
Return Codes	None

6.2.3 sysDuplexIntInstallHandler: Installing Interrupt Service Functions

In interrupt mode, this function installs <code>sysDuplexIntHandler</code> in the processor vector table, spawns the <code>sysDuplexDPRTask</code> function as a task, and creates a communication channel (for example, a message queue) between the two. In addition, it adds the S/UNI-DUPLEX device to a list of devices that need interrupt servicing.

In polling mode, this function spawns the sysDuplexDPRTask function. This function periodically polls the device for interrupts and services the interrupts. It also adds the S/UNI-DUPLEX device to a list of devices that need polling services.

Prototype	<pre>INT4 sysDuplexIntInstallHandler(DUPLEX duplex)</pre>	
Inputs	duplex: Pointer to device context information	
Outputs	None	
Return Codes	DPX_SUCCESS	
	DPX_ERR_INT_ALREADY	
	DPX_ERR_INT_INSTALL	



S/UNI-DUPLEX DRIVER MANUAL

6.2.4 sysDuplexIntRemoveHandler: Removing Interrupt Service Functions

In interrupt mode, this function removes the specified device from the list of devices that need interrupt processing. If this is the last active device, the function deletes the sysDuplexDPRTask and associated message queue. It also removes the sysDuplexIntHandler function from the processor's interrupt-vector table.

In polling mode, this function removes the specified device from the list of devices that need polling services. If this is the last active device, this function deletes sysDuplexDPRTask.

Prototype	VOID sysDuplexIntRemoveHandler(DUPLEX duplex)
Inputs	duplex: Pointer to device context information
Outputs	None
Return Codes	None

6.2.5 sysDuplexIntHandler: Calling duplexISR

In interrupt mode, this function calls duplexISR for each device with interrupt processing enabled. The driver calls this function when one or more S/UNI-DUPLEX devices interrupt the microprocessor. If duplexISR detects at least one valid pending interrupt condition, then this function queues the interrupt context information for later processing by sysDuplexDPRTask.

In polling mode, this function is not used.

Prototype sysDuplexIntHandler(INT4 Irq)

Inputs Irq: IRQ number of interrupt

Outputs None

Return Codes None



S/UNI-DUPLEX DRIVER MANUAL

6.2.6 sysDuplexDPRTask: Calling duplexDPR

In interrupt mode, the driver spawns this function as a separate task within the RTOS. It retrieves interrupt status information queued for it by the <code>sysDuplexIntHandler</code> function and calls the <code>duplexDPR</code> function for the appropriate device.

In polling mode, the driver spawns this function as a separate task within the RTOS. It periodically calls the duplexDPR function for each active device.

Prototype	VOID	<pre>sysDuplexDPRTask(VOID)</pre>
Inputs	None	
Outputs	None	
Return Codes	None	

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799



PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

7 PORTING DRIVERS

This section outlines how to port the S/UNI-DUPLEX device driver to your hardware and OS platform.

Note: Because each platform and application is unique, this manual can only offer guidelines for porting the S/UNI-DUPLEX driver.

7.1 Driver Source Files

The C source files listed in Figure 7-1 contain the code for the S/UNI-DUPLEX driver. You may need to modify the code or develop additional code. The code is in the form of constants, macros, and functions. For the ease of porting, the code is grouped into source files (src) and include files (inc). The src files contain the functions and the inc files contain the constants and macros.



Figure 10: Driver Source Files

7.2 Driver Porting Procedures

The following steps summarize how to port the S/UNI-DUPLEX driver to your platform. The following sections describe these steps in more detail.



S/UNI-DUPLEX DRIVER MANUAL

Note: Because each platform and application is unique, this manual can only offer guidelines for porting the S/UNI-DUPLEX driver.

To port the S/UNI-DUPLEX driver to your platform:

- 1. Port the driver's OS extensions (page 98):
 - Data types
 - OS specific services
 - Utilities and interrupt services that use OS specific services
- 2. Port the driver to your hardware platform (page 100):
 - Port the device detection function.
 - Port low-level device read-and-write macros.
 - Define hardware system-configuration constants.
- 3. Port the driver's application-specific elements (page 102):
 - Define the task-related constants.
 - Code the callback functions.
- 4. Build the driver (page 103).

7.2.1 Porting Driver OS Extensions

The OS extensions encapsulate all OS specific services and data types used by the driver. The dpx_rtos.h file contains data types and compiler-specific data-type definitions. It also contains macros for OS specific services used by the OS extensions. These OS extensions include:

- Task management
- Message queues
- Timers
- Events
- Semaphores





C. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Memory Management

In addition, you may need to modify functions that use OS specific services, such as utility and interrupt-event handling functions. The dpx_rtos.c file contains the utility and interrupt-event handler functions that use OS specific services.

To port the driver's OS extensions:

- 1. Modify the data types in dpx_rtos.h. The number after the type identifies the data-type size. For example, UINT4 defines a 4-byte (32-bit) unsigned integer. Substitute the compiler types that yield the desired types as defined in this file.
- 2. Modify the OS specific services in dpx_rtos.h. Redefine the following macros to the corresponding system calls that your target system supports:

Service Type	Macro Name	Description	
Memory	sysDuplexMemAlloc	Allocates the memory block	
	sysDuplexMemFree	Frees the memory block	
	sysDuplexMemCopy	Copies the memory block from src to dest	
Semaphore	sysDuplexSemCreate	Creates the mutually exclusive semaphore	
	sysDuplexSemDelete	Frees the mutually exclusive semaphore	
	sysDuplexSemGive	relinquishes the mutually exclusive semaphore	
	sysDuplexSemTake	Gets the mutually exclusive semaphore	

3. Modify the utilities and interrupt services that use OS specific services in the dpx_rtos.c. The dpx_rtos.c file contains the utility and interrupt-event handler functions that use OS specific services. Refer to the function headers in this file for a detailed description of each of the functions listed below:

DRIVER MANUAL PMC-990799



erra, lnc. **РМ7350 S**/

PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Service Type	Function Name	Description	
Memory	sysDuplexMemSet	Sets each character in the memory buffer	
	duplexGetIndBuf	Gets a block of memory for the indication buffer	
	duplexReturnIndBuf	Frees the indication buffer	
Timer	sysDuplexDelayFn	Sets the task execution delay in milliseconds	
Interrupt	sysDuplexIntInstallHandler	Installs the interrupt handler for the OS	
	sysDuplexIntRemoveHandler	Removes the interrupt handler from the OS	
	sysDuplexIntHandler	Interrupt handler for the S/UNI-DUPLEX device	
	sysDuplexDPRTask	Deferred interrupt-processing routine (DPR)	

7.2.2 Porting Drivers to Hardware Platforms

This section describes how to modify the S/UNI-DUPLEX driver for your hardware platform.

Before you build the driver, ensure that you port the driver's OS extensions (page 98).



S/UNI-DUPLEX DRIVER MANUAL

To port the driver to your hardware platform:

- Modify the device detection function in the dpx_hw.c file. The function sysDuplexDeviceDetect is implemented for a PCI platform. Modify it to reflect your specific hardware interface. Its purpose is to detect a S/UNI-DUPLEX device based on a user-context input parameter. It returns two output parameters:
 - The base address of the S/UNI-DUPLEX device
 - A pointer to the system-specific configuration information
- 2. Modify the low-level device read/write macros in the dpx_hw.h file. You may need to modify the raw read/write access macros (sysDuplexRawRead and sysDuplexRawWrite) to reflect your system's addressing logic.
- 3. Define the hardware system-configuration constants in the dpx_hw.h file. Modify the following constants to reflect your system's hardware configuration:

#define	Description	Default
DPX_MEM_ADDR_RANGE	The assigned address memory range for each S/UNI-DUPLEX device. Your system's memory map determines it.	0x800
DPX_ADAPTER_MAX_UNITS	The maximum number of S/UNI-DUPLEX cards allowed in the system	7
	Note: The DSLAM architecture allows up to 16 S/UNI-DUPLEX cards.	
DPX_ADAPTER_MAX_DEVS	The maximum number of S/UNI-DUPLEX devices on each card	1



S/UNI-DUPLEX DRIVER MANUAL

7.2.3 Porting Driver Application-Specific Elements

Application specific elements are configuration constants used by the API for developing an application. This section describes how to modify the application specific elements in the S/UNI-DUPLEX driver.

Before you port the driver's application-specific elements, ensure that you:

- 1. Port the driver's OS extensions (page 98).
- 2. Port the driver to your hardware platform (page 100).

To port the driver's application-specific elements:

1. Define the following driver task-related constants for your OS-specific services in file dpx_rtos.h:

#define	Description	Default
DPX_DPR_TASK_PRIORITY	Deferred Task (DPR) task priority	85
DPX_DPR_TASK_STACK_SZ	DPR task stack size, in bytes	4096
DPX_POLLING_DELAY	Constant used in polling task mode, this constant defines the interval time in millisecond between each polling action	10
DPX_TASK_SHUTDOWN_DELAY	Delay time in millisecond. When clearing the DPR loop active flag in the DPR task, this delay is used to gracefully shutdown the DPR task before deleting it.	10
DPX_MAX_DPR_MSGS	The queue message depth of the queue used for pass interrupt context between the ISR task and DPR task	10
DPX_MAX_IND_BUFSZ	Maximum indication buffer size in bytes	53
DPX_MAX_NUM_DEVS	The maximum number of S/UNI-DUPLEX devices in the system (from 1 to 128)	7

DRIVER MANUAL PMC-990799 PMC

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

- 2. Code the callback functions according to your application. There are four sample callback functions in the dpx_test.c file. You can use these callback functions or you can customize them before using the driver. The driver will call these callback functions when an event occurs on the device. These functions must conform to the following prototypes:
 - VOID indDuplexNotify(DPX_USR_CTXT usrCtxt, sDPX_IND_BUF *psIndCtxt)
 - VOID indDuplexRxBOC(DPX_USR_CTXT usrCtxt, sDPX_IND_BUF *psIndCtxt)
 - VOID indDuplexCell(DPX_USR_CTXT usrCtxt, sDPX_IND_BUF *psIndCtxt)
 - UINT1 pCellTypeFn(UINT1 *pulHdr, UINT4 *pu4Crc32Prev)

7.2.4 Building Drivers

This section describes how to build the S/UNI-DUPLEX driver.

Before you build the driver, ensure that you:

- 1. Port the driver's OS extensions (page 98).
- 2. Port the driver to your hardware platform (page 100).
- 3. Port the driver's application-specific elements (page 102).

To build the driver:

- 1. Modify the makefile's compile-switch flag DPX_CSW_INTERRUPT_MODE. Set it to one for interrupt mode or zero for polling mode.
- 2. Set the makefile's compile-switch flag CSW_PV_FLAG to zero. This disables the test code specific to product verification.
- 3. Ensure that the directory variable names in the makefile reflect your actual driver and directory names.
- 4. Compile the source files and build the S/UNI-DUPLEX API driver library using your make utility.

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

5. Link the S/UNI-DUPLEX API driver library to your application code.



S/UNI-DUPLEX DRIVER MANUAL

APPENDIX: CODING CONVENTIONS

This section describes the coding and naming conventions used in the implementation of the driver software. This section also describes the variable types.

Definition of Variable Types

The following table describes the variable types used by the S/UNI-DUPLEX driver.

Туре	Description
UINT1	unsigned integer, 1 byte
UINT2	unsigned integer, 2 bytes
UINT4	unsigned integer, 4 bytes
INT1	signed integer, 1 byte
INT2	signed integer, 2 bytes
INT4	signed integer, 4 bytes
VOID	void
DPX_USR_CTXT	void *, pointer to user maintained device context
DUPLEX	void *, pointer to driver maintained device context

Table 16: Definition of Variable Types

Naming Conventions

The names for variables, functions, and macros (but not constants) include prefixes that indicate their type. Variable, function, and macro names that contain multiple words have the first letter of each word capitalized.

Variables

The following table describes the prefixes used for the driver's variables.





S/UNI-DUPLEX DRIVER MANUAL

Table 17: Variable Naming Conventions

Variable Type	Prefix	Example	
UINT1	ul	ulFlag	
UINT2	u2	u2Code	
UINT4	u4	u4Val	
INT1	il	ilFlag	
INT2	i2	i2Code	
INT4	i4	i4Val	
Structure Variable	S	sCellHdr	
Enumerated Type	е	eHssRegId	
Pointers	р	pulFlag	
		pi4Val	
		psCellHdr	
		peHssRegId	
Pointer to a Pointer	qq	ppulFlag	
		ppi4Val	
		ppsCellHdr	
		ppeHssRegId	

Functions and Macros

The following table describes the prefixes used for the driver's functions and macros.

DRIVER MANUAL PMC-990799



ierra, Inc. PM7350 S

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Table 18: Function and Macro Naming Conventions

Function Type	Prefix	Example Name
API Functions	duplex	duplexAdd
Indication Functions	indDuplex	indDuplexRxCell
System-Specific Functions and Macros	sysDuplex	sysDuplexIntHandler

Definable Constants

You can define some constants using the "#define" command. These constants have names that are composed of all uppercase letters with underscores separating multiple words. An example is DPX_NUM_HSS_LNKS.

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL
DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

ACRONYMS

- API: Application programming interface
- DDB: Device data block
- BOC: Bit oriented code
- DPR: Deferred interrupt-processing routine
- GDD: Global driver database
- HCS: Header check sequence
- HSS link: High-speed serial link
- ISR: Interrupt service routine
- RTOS: Real-time operating system

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799

INDEX

Α

Accessing Registers, 89 Accumulating Counts, 76, 77, 78 Acronyms, 109 Activating Devices, 52 Adding Devices, 45 addr, 90 Addresses, 90 Allocating Memory, 84 API Module, 17 Application Interface Functions, 43 Architecture, 16, 17 Auto Reset, 73

в - С

Base Addresses, 90 BOC, 15, 43, 65, 66, 67 Buffers, 85, 86 Building Drivers, 103 Callback Functions, 64 Calling duplexDPR, 25, 95 Calling duplexISR, 24, 91, 94 Cell Data Structures, 29 Cell Extraction, 22, 23, 59 Cell Insertion, 59, 60 Cell-Control Data Structure, 29 Cell-Header Data Structure, 29 CELLXFERRE, 37 Clocked Serial-Data Interface Functions, 71 Clocks, 55 Coding Conventions, 105 Collecting Statistics, 16, 76 Configuration Information, 57, 58, 68 Configuring HSS Links, 70 Contents of Extract-FIFO-Ready Registers, 63 Count Structures, 38 Count Clock Lock Fail, 39 Count Extract Cell CRC Error, 39 Count Extract Cells, 41 Count Interrupts, 41 Count_Invalid_SOC_Sequence, 39 Count Micro Insert Fifo Full, 39 Count Micro Insert Fifo Ready, 39 Count_Phy_Input_Cell_Xfered, 39 Count Phy Input Parity, 39

PMC PMC-Sierra, Inc.

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

PM7350 S/UNI-DUPLEX DRIVER

Count Phy Output Error, 39 Count Rx BOCs, 41 Count_Rx_Lc_Fifo_Overflow, 39 Count RxHss Active Bit, 40 Count RxHss Count Overflow, 41 Count RxHss Count Updated, 41 Count RxHss Extract Fifo, 40 Count RxHss HCS Error, 41 Count_RxHss_In_Delin, 40 Count RxHss In Sync, 41 Count RxHss Loss Of Signal, 40 Count_RxHss_No_Active_Bit, 40 Count_RxHss_Out_Of_Delin, 40 Count RxHss Out Of Sync, 41 Count_RxHss_Signal_Detected, 40 Count RxSerChnl Fifo Overflow, 40 Count RxSerChnl HCS Error, 40 Count_RxSerChnl_In_Delin, 39 Count RxSerChnl in Sync, 40 Count RxSerChnl Out Of Delin, 39 Count RxSerChnl Out Of Sync, 40 Count Tx Hss Count Overflow, 39 Count Tx Hss Count Updated, 39 Count_Tx_Lc_Fifo_Overflow, 39 Counts, 38, 39 Counts for All Cells, 78 Counts for Received Cells, 76 Counts for Transmitted Cells, 77 Creating Semaphores, 87 CSW_PV_FLAG, 103

D

Data Structures, 29 DDBs, 33, 34 Deactivating Devices, 52 Deallocating Memory, 85 Deferred-Processing Routine Module, 19 Delaying Functions, 86 Deleting Devices, 45, 47 Deleting Semaphores, 87 dest, 99 Device Activation, 52 Device Data Block, 18, 34 Device Deactivation, 52 Device Deletion, 45

DRIVER MANUAL PMC-990799

ISSUE 1

Device Diagnostics, 15, 53 Device Initialization, 15, 21, 22, 49 **Device Interface Functions**, 89 Device Re-initialization, 21, 22 Device Reset, 45 Device Shutdown, 21, 22 **Device-Configuration Data Structures**, 30 Device-Context Data Structures, 33 Diagnosing Devices, 15, 53 Diagnostic or Line Loopback, 54 Disabling Diagnostic or Line Loopback, 54 DPR, 19 DPR Buffers, 85, 86 DPX_ADAPTER_MAX_DEVS, 101 DPX_ADAPTER_MAX_UNITS, 101 DPX ALL HSS REGS, 58, 59 DPX ALL PHY REGS, 69, 70 DPX_CB_NOTIFY, 51 DPX CB RX BOC, 51 DPX CB RX CELL, 51 DPX CELLTYPE FN, 31, 36 DPX CLK BIT SER, 35 DPX_CLK_SER_LCD_CNT_THRESH, 72, 73 DPX CLK SER RX CHNL CFG, 72, 73 DPX CLK SER RX HCS ERR CNT, 72 DPX_CLK_SER_RX_INT_ENBLS, 72, 73 DPX CLK SER RX INT STATUS, 72 DPX CLK SER TX DATA, 74, 75 DPX_CLK_SER_TX_SER_FRM_BIT_THRESH, 74, 75 DPX_CSW_INTERRUPT_MODE, 103 DPX DEVICE NOT DETECTED, 91 DPX DIAG LPBK, 54 DPX_DPR_TASK_PRIORITY, 102 DPX DPR TASK STACK SZ, 102 DPX EMPTY, 35, 46, 47, 54, 55, 79 DPX EOM FN, 65 DPX_ERR_CB_FN_NOT_INSTALLED, 63 DPX ERR CELL RX CRC, 63 DPX_ERR_CELL_TX_BUSY, 61 DPX ERR DEV ID TYPE, 46 DPX ERR DEV NOT DETECTED, 46 DPX_ERR_EXCEED_REG_RANGE, 48, 49 DPX ERR INDIRECT CHANNEL BUSY, 50 DPX ERR INT ALREADY, 93 DPX_ERR_INT_INSTALL, 93 DPX ERR INVALID BOC, 66, 67 DPX ERR INVALID CB TYPE, 51, 52 DPX_ERR_INVALID_CHNL_ID, 72, 73, 75, 76

S/UNI-DUPLEX DRIVER MANUAL

DPX ERR INVALID FLAG, 55 DPX ERR INVALID HSS ID, 55 DPX_ERR_INVALID_INIT_VECTOR, 50 DPX ERR INVALID LNK ID, 63, 66, 67, 68, 77 DPX_ERR_INVALID_LPBK_TYPE, 55 DPX ERR MEM ALLOC, 45, 46 DPX ERR MODULE ALREADY INIT, 45 DPX FAILURE, 54 DPX IND CB FN, 31, 35, 36, 50 DPX LINE LPBK, 54 DPX_LPBK_RESET, 54 DPX LPBK SET, 54 DPX MAX DPR MSGS, 102 DPX_MAX_IND_BUFSZ, 102 DPX MAX NUM DEVS, 34, 38, 102 DPX MEM ADDR RANGE, 101 DPX NUM HSS LNKS, 107 DPX POLLING DELAY, 102 DPX_RX_HSS_CELL_FILTER_CFG_RXD, 58, 59 DPX RX HSS CFG RXD, 58, 59 DPX RX HSS LNK SELECT AUTO, 56, 57 DPX_RX_HSS_LNK_SELECT_RXD, 56, 57 DPX RXD, 54, 62, 66, 67, 68, 77 DPX_SCI_ANY_PHY_EXT_ADDR_MASK, 69, 70 DPX_SCI_ANY_PHY_EXT_ADDR_MATCH, 69, 70 DPX SCI ANY PHY ICA ENBL , 69, 70 DPX SCI ANY PHY ICA ENBL LSB, 69, 70 DPX SCI ANY PHY ICA ENBL MSB, 69, 70 DPX_SCI_ANY_PHY_INP_CFG_, 69, 70 DPX SCI ANY PHY OUT ADDR MATCH, 69,70 DPX_SCI_ANY_PHY_OUT_CFG, 69, 70 DPX SCI ANY PHY OUT POLL RNG, 69, 70 DPX_SCI_ANY_SLAVE, 35 DPX SCI MASTER, 35 DPX SEM ID, 36 DPX_TASK_SHUTDOWN_DELAY, 102 DPX TX HSS CFG, 58, 59 DPX USR CTXT, 34, 46, 90, 103, 105 Driver Data Structures, 29 Driver Functions and Features, 15 Driver Hardware-Interface Module, 18 Driver Initialization, 44 Driver Library Module, 18 Driver Shutdown, 44, 45 Driver Software States, 19, 20

DRIVER MANUAL PMC-990799 PMC PMC-Sierra, Inc.

PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Driver Source Files, 97 **DSLAM**, 101 duplexActivate, 52 duplexAdd, 45, 46, 90, 107 duplexCheckExtractFifos, 63, 81 duplexDeactivate, 52, 53 duplexDelete, 25, 43, 47, 48 duplexDPR, 19, 23, 25, 26, 43, 49, 50, 51, 80, 81, 91, 92, 93, 95 duplexEnableRxCellInd, 64 duplexExtractCell, 61, 62, 64, 81 duplexGetAllHssCounts, 78 duplexGetAllHssLnkCounts, 76, 77 duplexGetClockStatus, 55 duplexGetHssLnkRxCounts, 76, 78 duplexGetHssLnkTxCounts, 77, 78 duplexGetIndBuf, 85, 100 duplexGetStatisticCounts, 79 duplexHssActiveLnkGetCfg, 56 duplexHssActiveLnkSetCfg, 57 duplexHssGetConfig. 57 duplexHssSetConfig, 58, 59 duplexInit, 49, 50 duplexInsertCell, 60 duplexInstallCellTypeFn, 64, 65 duplexInstallIndFn, 50 duplexISR, 19, 23, 24, 25, 26, 91, 92, 94 duplexLoopback, 54 duplexModuleInit, 44 duplexModuleShutdown, 45 duplexRead, 48 duplexRegisterTest, 53 duplexRemoteReset, 47 duplexRemoveIndFn, 51 duplexReset, 46, 47 duplexResetStatisticCounts, 79 duplexReturnIndBuf, 80, 81, 86, 100 duplexRxBOC, 66 duplexRxSerChnIHCSCntResetEn, 73 duplexRxSerChnlReadReg, 71 duplexRxSerChnlWriteReg, 72 duplexSciAnyPhyGetConfig, 68 duplexSciAnyPhySetConfig, 70 duplexSetAutoRDITx, 67 duplexTxBOC, 65 duplexTxSerChnlReadReg, 74 duplexTxSerChnlWriteReg, 75 duplexWrite, 49

Ε

eCbType, 50, 51 eClkSerRegId, 71, 72, 73, 74, 75 eDevMode, 35 eDevState, 35 eDPX_CB_TYPE, 50, 51 eDPX_CLK_SER_REG, 71, 72, 74, 75 eDPX_HSS_REG, 57, 59 eDPX MODE, 35 eDPX SCI ANY PHY REG, 68, 70 eDPX_STATE, 35 eHSS LNK SEL, 56, 57 eHssRegId, 57, 58, 59, 106 eLnkSel, 57 Enabling Auto Reset of HCS Error Registers, 73 Enabling Diagnostic or Line Loopback, 54 Enabling Received Cell Indicators, 64 Error Registers, 73 errored, 16, 76, 77, 78 eSciAnyPhyRegId, 68, 69, 70 Events, 80 Extract-FIFO-Ready Register, 63 Extracting Cells, 22, 23, 61

F - G

FIFOs, 16, 31, 36, 63, 81 Files, 97, 103 Flow Processing, 20 FOVRE, 37 Functions and Features, 15 GDD, 33, 44, 45, 109 Getting Contents of Extract-FIFO-Ready Registers, 63 Getting Device Base Addresses, 90 Getting DPR Buffers, 85 Getting HSS-Link Configuration Information, 57 Getting HSS-Link Selection Method Information, 56 Getting SCI-PHY/Any-PHY Configuration Information, 68 Giving Semaphores, 87 Global Driver-Database Structure, 33

H - K

Hardware Interface, 18, 89 HCS, 38, 71, 73, 76, 78, 109 HCSPASS, 33 HSS Counts, 38 HSS Link Cell Extraction, 59

DRIVER MANUAL PMC-990799

ISSUE 1

HSS Link Cell Insertion, 59 HSS Link Configuration, 15, 56, 57, 58 HSS Link Selection, 56 HSS Links, 57, 60, 61, 70 IDLEI, 66 IFCLK, 55 inc, 13, 97 indDuplex, 107 indDuplexCell, 103 indDuplexNotify, 80, 103 indDuplexRxBOC, 66, 80, 103 indDuplexRxCell, 81, 107 Indication Callbacks, 16, 50, 51, 80 indNotify, 31, 35 indRxBOC, 31, 36 indRxCell, 31, 36 Init, 20 Initialization, 44 Initialization Data Structure, 30 Initializing Devices, 15, 21, 22, 49 Initializing Driver Modules, 44 Inserting Cells into HSS Links, 60 Installing Callback Functions, 64 Installing Indication Callback Functions, 50 Installing Interrupt Service Functions, 93 Interrupt Data Structures, 36 Interrupt Processing, 92 Interrupt Service Functions, 93, 94 Interrupt Servicing, 16, 23, 91 Interrupt Statuses, 92 Interrupt-Context Data Structure, 37 Interrupt-Enable Data Structure, 36 Interrupt-Service Routine Module, 19 IRQ, 34, 94 ISR, 16, 23, 36, 91 ISR Data Structure, 36 ISR Module, 19

L - M

Library Module, 18 Line Loopback, 54 lockId, 36 loopback, 15, 53, 54, 55, 65, 66, 67, 81 LSB, 32 Make Files, 103 malloc. 84 Memory, 84, 85 Modifying HSS-Link Configuration Information, 58

S/UNI-DUPLEX DRIVER MANUAL

Monitoring Device Clocks, 55 MSB, 32

PMC-Sierra, Inc.

N - Q nbytes, 84 Notifying Applications, 80, 81 OFCLK, 55 Other Devices, 47 pBuf, 86 pCbFn, 50, 51 pCellTypeFn, 31, 36, 65, 103 PCI, 34, 101 pDdb, 34 pDuplex, 46 peHssRegId, 106 peLnkSel, 56 pIndBuf, 80, 81 Polling Servicing, 26 Porting Application-Specific Elements, 102 Porting Drivers, 97, 100 Porting Drivers to Hardware Platforms, 100 Porting OS Extensions, 98 Porting Procedures, 97 Porting Quick Start, 13 ppeHssRegId, 106 ppsCellHdr, 106 ppSysInfo, 90, 91 Processing Flows, 20 Processing Interrupts, 92 psCellHdr, 60, 61, 62, 106 psCtrl. 60, 61, 62 psHssCnts, 78 psHssRegs, 57, 58, 59 psIndCtxt, 103 psSciAnyPhyRegs, 68, 69, 70 psStatCounts, 79 pSysInfo, 34

RDI, 65, 66, 67, 68, 81 RDIDIS, 65 Reading from Receive Serial-Channel Context Bytes, 71 Reading from Received BOC, 66 Reading from Register Address Locations, 90 Reading from Registers, 48 Reading from Transmit Serial-Channel Context Bytes, 74 Ready Extract-Cell-FIFOs, 81

DRIVER MANUAL PMC-990799 PMC PMC-Sierra, Inc.

PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

Receive Serial-Channel Context Bytes, 71, 72 Received BOC, 66, 80 Received Cell Indicator, 64 Received Cells, 76 REFCLK, 55 Register Access, 53, 89 Register Address Locations, 90 Register Data Structure, 31 Registering Interrupt Statuses, 92 Registers, 48, 49, 73 Re-initializing Devices, 21, 22 Remote Defect Indications, 67 Removing Indication Callback Functions, 51 **Removing Interrupt Service Functions**, 94 Resetting Devices, 45, 46 Resetting Other Devices, 47 Resetting Statistical Counts, 79 **Retrieving Statistical Counts, 79** Returning DPR Buffers, 86 ROOLE, 37 ROOLV, 37 RSTOB, 47 **RTOS**, 25 RTOS Interface, 18, 83

S

sCellHdr, 106 SCI-PHY/Any-PHY Configuration, 68 sClkSerRegs, 32 sDPX_CELL_CTRL, 30, 60, 62 sDPX CELL HDR, 29, 60, 62 sDPX_CLK_SER_REGS, 32, 33 sDPX_DDB, 34 sDPX GDD, 34 sDPX HSS CNTS, 38, 78 sDPX HSS REGS, 32, 57, 59 sDPX_IND_BUF, 80, 81, 85, 86, 103 sDPX_INIT_VECT, 31, 35, 50 sDPX INIT VECTOR, 31, 35 sDPX INT CTXT, 38 sDPX_INT_ENBLS, 32, 35, 37 sDPX REGS, 31 sDPX SCI ANY PHY REGS, 32, 68, 70 sDPX_STAT_COUNTS, 36, 39, 79 Selection Methods, 56 Semaphores, 86, 87 semDelete, 87 semGive, 88 semId, 87, 88

semMCreate, 87 semTake, 87 Setting Active HSS Links, 57 sHssRegs, 32 Shutdown, 44 Shutting Down Devices, 21, 22 Shutting Down Drivers, 45 Significant Events, 80 sInitVector, 35, 50 sIntEnbls, 35 sIntEnRegs, 32 Software States, 19, 20 Source Files, 97 src, 13, 97, 99 sRegInfo, 31 sSciAnyPhyRegs, 32 sStatCounts, 36 States, 19, 20 Statistical Counts, 38, 39, 79 Statistics Collection, 16, 76 sysDuplex, 26, 87, 99, 107 sysDuplexDelayFn, 86, 100 sysDuplexDeviceDetect, 90, 101 sysDuplexDPR, 23 sysDuplexDPRTask, 25, 26, 27, 91, 93, 94, 95, 100 sysDuplexIntHandler, 24, 25, 26, 91, 92, 93, 94, 95, 100, 107 sysDuplexIntInstallHandler, 25, 26, 92, 93, 100 sysDuplexIntRemoveHandler, 92, 94, 100 sysDuplexISR, 23 sysDuplexMemAlloc, 84, 85, 99 sysDuplexMemCopy, 99 sysDuplexMemFree, 85, 99 sysDuplexMemSet, 100 sysDuplexRawRead, 48, 90, 101 sysDuplexRawWrite, 49, 90, 101 sysDuplexSemCreate, 87, 99 sysDuplexSemDelete, 87, 99 sysDuplexSemGive, 87, 88, 99 sysDuplexSemTake, 87, 99

T - Z

Taking Semaphores, 87 Timer Operations, 86 Transmit Serial-Channel Context Bytes, 74, 75 Transmitted Cells, 77 Transmitting BOC, 65 Transmitting Remote-Defect Indications, 67

DRIVER MANUAL PMC-990799 PMC PMC-Sierra, Inc.

C. PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

UDF, 61 USR_CTXT, 80, 81 usrCtxt, 34, 46, 80, 81, 90, 103 val, 90 Verifying Register Access, 53 Writing to Receive Serial-Channel Context Bytes, 72 Writing to Registers, 48, 49, 90 Writing to Transmit Serial-Channel Context Bytes, 75

DRIVER MANUAL PMC-990799



ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

DRIVER MANUAL PMC-990799 PMC PMC-Sierra, Inc.

PM7350 S/UNI-DUPLEX DRIVER

ISSUE 1

S/UNI-DUPLEX DRIVER MANUAL

CONTACTING PMC-SIERRA, INC.

PMC-Sierra, Inc. 105-8555 Baxter Place Burnaby, BC Canada V5A 4V7

Tel: (604) 415-6000 Fax: (604) 415-6200

Document Information: document@pmc-sierra.com Corporate Information: info@pmc-sierra.com Application Information: apps@pmc-sierra.com Web Site: http://www.pmc-sierra.com

None of the information contained in this document constitutes an express or implied warranty by PMC-Sierra, Inc. as to the sufficiency, fitness or suitability for a particular purpose, merchantability, performance, compatibility with other parts or systems, of any of the products of PMC-Sierra, Inc., or any portion thereof, referred to in this document. PMC-Sierra, Inc. expressly disclaims all representations and warranties of any kind regarding the contents or use of the information, including, but not limited to, express and implied warranties of accuracy, completeness, merchantability, fitness for a particular use, or non-infringement.

In no event will PMC-Sierra, Inc. be liable for any direct, indirect, special, incidental or consequential damages, including, but not limited to, lost profits, lost business or lost data resulting from any use of or reliance upon the information, whether or not PMC-Sierra, Inc. has been advised of the possibility of such damage.

© 1999 PMC-Sierra, Inc.

PMC-990799 (P1), ref PMC-981033 (P2)

Issue date: July 1999